▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación

10

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación 10

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.
Gerente:
ANTONIO G. CUERPO.
Directora de producción:
MARIA LUISA SUAREZ PEREZ.
Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.
Diseño y maquetación:
BRAVO-LOFISH.
Fotografía:
EQUIPO GALATA.
Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectúra. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomuniación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanero, Licenciada en Informática. APLICACIONES: Soledad Tamariz, Diplomada en Telecomunicación. OTROS LENGUAJES (sistemas operativos): Domingo Villaseñor, Diplomado en Informática, y Lenguaje C: Enrique Serrano, Ingeniero en Telecomunicación.

Ediciones Siglo Cultural, S.A.
Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.
Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.
Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.
Distribución en Ecuador: Muñoz Hnos.
Distribución en Perú: DISELPESA.
Distribución en Chile: Alfa Ltda.
Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-089-8
ISBN de la obra: 84-7688-068-7
Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.
Imprime:
MATEU CROMO. Pinto (Madrid).

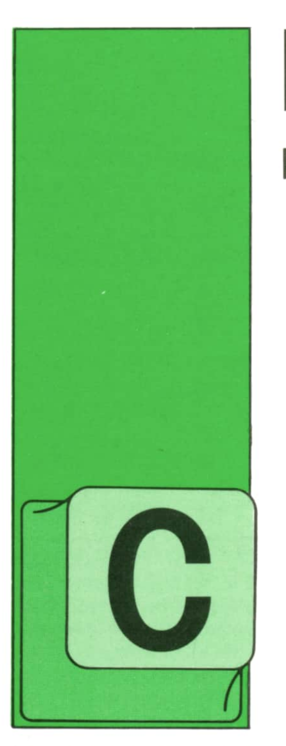
Depósito legal: M-5-677-1987
Printed in Spain - Impreso en España.
Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.
Mayo, 1987.
P.V.P. Canarias: 335,–.

INDICE

INFORMATICA BASICA

REPRESENTACION INTERNA DE LOS DATOS

Sistemas de numeración



COMO ya sabemos, el ordenador trabaja internamente con datos binarios (cadenas de unos y ceros), por medio de los cuales representa números, nombres, etcétera. Existen varios formatos internos de representación de los distintos tipos de datos, aunque en la mayoría de los casos el programador trabaja en un *lenguaje de programación* lejano a la máquina y, por ello, no necesita considerar el formato interno de los datos que maneja.

El hombre trabaja generalmente en sistema decimal, y el ordenador en sistema binario; en ambos la representación de los números se realiza mediante cadenas de símbolos, dependiendo el valor de cada símbolo de la posición que ocupe dentro de la cadena.

Un número decimal ordinario, de los que todos usamos, consta de un conjunto de dígitos decimales y, posiblemente de una coma decimal. La forma general y su interpretación se muestran en la figura 1.

Número binario	Expresión	Número decimal
0	0	0
1	1	1
10	1 × 2 + 0	2
11	1 × 2 + 1	3
100	1 × 4 + 0 × 2 + 0	4
101	1 × 4 + 0 × 2 + 1	5
110	1 × 4 + 1 × 2 + 0	6
111	1 × 4 + 1 × 2 + 1	7
1000	1 × 8 + 1 × 4 + 0 × 2 + 0	8
1001	1 × 8 + 0 × 4 + 0 × 2 + 1	9
1010	1 × 8 + 0 × 4 + 1 × 2 + 0	10
1011	1 × 8 + 0 × 4 + 1 × 2 + 1	11
:	:	:

La elección de 10 como *base de exponenciación* se hace a causa de que utilizamos números decimales o en base 10. Cuando tratamos con ordenadores es mucho mejor utilizar otras bases distintas de 10. Las bases más importantes son 2, 8, 16. Los sistemas de numeración correspondientes a estas bases se llaman *binario, octal y hexadecimal.*

Sistema binario

El conjunto de símbolos utilizados en este sistema de numeración se limita a dos (0,1); en consecuencia, la única forma de representar un número binario es mediante una cadena de dígitos binarios, también llamados *bits:* ceros y unos.

Este sistema de numeración, ideado por Leibniz en el siglo XVII, constituye el alfabeto interno de los ordenadores electrónicos.

En este sistema, al tener sólo dos símbolos, un valor 2 habrá que representarlo como 10, ya que el símbolo 2 no existe en binario. En este sistema, un símbolo en la segunda posición tendrá un valor dos veces superior (base =2) al de los símbolos de la primera posición; un símbolo en la tercera posición (lo que conocemos en decimal como centena), un valor dos veces superior al de aquéllos en segunda posición, etc.

Sistema octal

En este sistema sólo son válidos los símbolos:

0, 1, 2, 3, 4, 5, 6, 7

Por tanto, el peso de cada uno de los dígitos de un número será 8^{n-1}, siendo *n* la posición que ocupa si contamos de derecha a izquierda. Veremos esto más adelante en la conversión entre sistemas.

Decimal	Binario	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14
30	11110	36	1E
40	101000	50	28
50	110010	62	32
60	111100	74	3C
70	1000110	100	46
80	1010000	120	50
90	1011010	132	5A
100	1100100	144	64
200	11001000	310	C8
300	100101100	454	12C
400	110010000	620	190
500	111110100	764	1F4
600	1001011000	1130	258
700	1010111100	1274	2BC
800	1100100000	1440	320
900	1110000100	1604	384
1000	1111101000	1750	3E8
2989	101110101101	5655	BAD



Números decimales y sus equivalentes a los sistemas binario, octal y hexadecimal.

Sistema hexadecimal

Este sistema está basado en un conjunto de 16 elementos;

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Como sólo tenemos diez dígitos numéricos diferentes se utilizan las letras, cuyo valor corresponde:

A=10, B=11, C=12, D=13, E=14, F=15

Para utilizar valores mayores se combinan estos símbolos básicos, de forma que para representar el número 16 pondríamos 10, etc.



Conversión de una base a otra

1. *Conversión de cualquier sistema a base decimal*

Basta sólo con aplicar la siguiente fórmula:

$$N_{(10} = d_n{}^*B_n + d_{n-1}{}^*B_{n-1} + \ldots + d_2{}^*2 + d_1{}^*B\ d_0$$

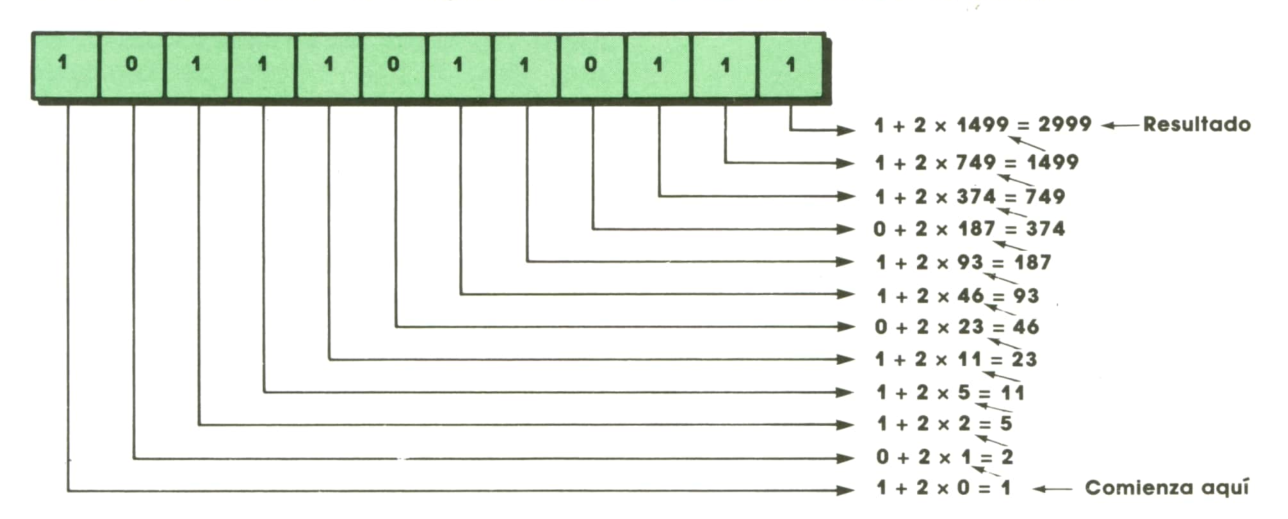
en donde d_0, d_1,...., d_n, son las cifras del número inicial, B es el número de símbolos del sistema de numeración (base del sistema) y $N_{(10}$ es la representación del número en sistema decimal.

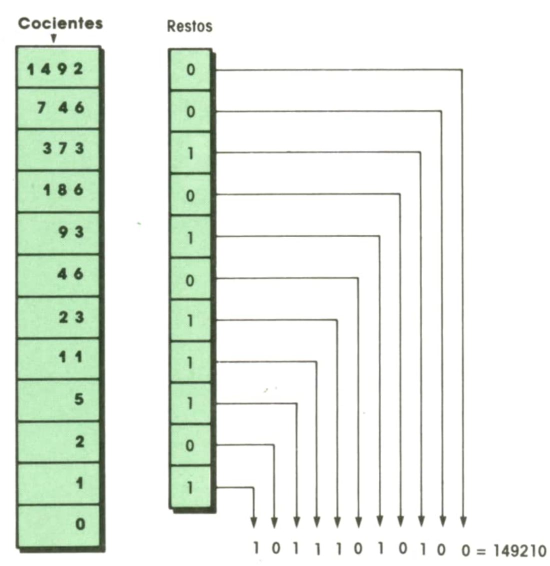
2. *Conversión de un número decimal a otro sistema*

Mucho más fácil, basta con dividir sucesivamente el número en base decimal por la nueva base; el cociente y los restos obtenidos (en orden contrario al que han aparecido) coinciden con el número expresado en el nuevo sistema.



Conversión del número binario 101110110111 a decimal.






Conversión del número 1492 a binario por divisiones sucesivas.

3. *Conversión binario-octal y binario-hexadecimal*

— Para convertir un número expresado en base dos a octal, se van agrupando los dígitos en grupos de tres. Cada uno de los grupos corresponde a un símbolo en base ocho.

— Para cambiar de base dos a base dieciséis, agrupamos los dígitos binarios en grupos de cuatro y hacemos lo mismo que en el caso anterior.

CONVERSION BINARIO-OCTAL

Binario	Octal
00	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

CONVERSION BINARIO-HEXADECIMAL

Binario	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F



Tipos de datos

Un dato es cualquier elemento de información que va a ser procesado por un ordenador. Cuando un programa va a trabajar con un gran número de datos, se facilitará mucho el trabajo si se organiza de forma apropiada su almacenamiento. Vamos a ver a continuación algunas formas de estructurar los datos:

1. Datos simples

Son aquéllos que no pueden descomponerse en partes significativas más simples. Los datos pueden o no alterarse durante la ejecución de programas. Si no se alteran reciben el nombre de CONSTANTES y, en caso contrario, de VARIABLES. Cada uno de estos datos tiene reservado en memoria el espacio suficiente para registrar su valor.

Dentro de este grupo de datos podemos hacer una división respecto al tipo de dato que estemos utilizando:

Datos numéricos

Para operar con un número se debe conocer de él:

— Su signo.

— El valor de cada una de sus cifras, en su orden.

— El lugar de la coma decimal en la cadena de cifras.

Por este motivo, el ordenador deberá representar todo ello en memoria al almacenar un dato numérico. La forma de hacerlo va de acuerdo al sistema de representación utilizado.

a) Coma fija.

El nombre viene de la forma de representar la posición de la coma decimal. El lugar viene predeterminado por el sistema. Existen tres formas de almacenamiento en coma fija:

— Binario puro.

— Decimal empaquetado.

— Decimal desempaquetado.

b) En coma flotante, para una base dada, los números se representan por medio de una *mantisa* y un *exponente*. De esta forma, el ordenador debe almacenar:

— El valor de la mantisa y su signo.

— El valor del exponente y su signo.

— El valor de la base que va implícito en el sistema elegido.

La coma flotante puede representarse en dos modos:

— Simple precisión.
— Doble precisión.

Datos alfabéticos y alfanuméricos

Este tipo de datos se almacena siempre en formato desempaquetado, es decir, un carácter por byte, ya que en este caso sí son necesarios más de cuatro bits para representar cada carácter. La configuración de ceros y unos que representa a cada carácter dependerá del *código* utilizado por el ordenador. En equipos de 8 bits los más utilizados son EBCDIC y ASCII.

CODIGO ALFANUMERICO ASCII											
DEC	**ASCII**	**HEX**	**DEC**	**ASCII**	**HEX**	**DEC**	**ASCII**	**HEX**	**DEC**	**ASCII**	**HEX**
0	NUL	00	32	SPACE	20	64	a	40	96	'	60
1	SOH	01	33	!	21	65	A	41	97	a	61
2	STX	022	34	d	22	66	B	42	98	b	62
3	ETX	03	35	#	23	67	C	43	99	c	63
4	EOT	04	36	$	24	68	D	44	100	d	64
5	ENQ	05	37	%	25	69	E	45	101	e	65
6	ACK	06	38	&	26	70	F	46	102	f	66
7	BEL	07	39		27	71	G	47	103	g	67
8	BS	08	40	(	28	72	H	48	104	h	68
9	HT	09	41	)	29	73	I	49	105	i	69
10	LF	0A	42	·	2A	74	J	4A	106	j	6A
11	VT	0B	43	+	2B	75	K	4B	107	k	6B
12	FF	0C	44	,	2C	76	L	4C	108	l	6C
13	CR	0D	45	–	2D	77	M	4D	109	m	6D
14	SO	0E	46	.	2E	78	N	4E	110	n	6E
15	SI	0F	47	/	2F	79	O	4F	111	o	6F
16	DLE	10	48	0	30	80	P	50	112	p	70
17	DC1	11	49	1	31	81	Q	51	113	q	71
18	DC2	12	50	2	32	82	R	52	114	r	72
19	DC3	13	51	3	33	83	S	53	115	s	73
20	DC4	14	52	4	34	84	T	54	116	t	74
21	NAL	15	53	5	35	85	U	55	117	u	75
22	SYN	16	54	6	36	86	V	56	118	v	76
23	ETB	17	55	7	37	87	W	57	119	w	77
24	CAN	18	56	8	38	88	X	58	120	x	78
25	EM	19	57	9	39	89	Y	59	121	y	79
26	SUB	1A	58	:	3A	90	Z	5A	122	z	7A
27	ESC	1B	59	;	3B	91	[	5B	123	{	7B
28	FS	1C	60	<	3C	92	\	5C	124	|	7C
29	GS	1D	61	=	3D	93	]	5D	125	}	7D
30	RS	1E	62	>	3E	94	(↑)	5E	126	DEL	7E
31	US	1F	63	?	3F	95	–(–)	5F	127		7F

2. Punteros

Son datos cuyo valor indica la dirección de memoria donde se encuentra otro dato. Por tanto, es una forma indirecta de acceder a la información.

Pueden ser útiles, por ejemplo, para encadenar informaciones en un cierto orden.





Los punteros son datos que no contienen más información que la de la dirección de memoria donde se encuentra otro dato. En este caso los punteros son campos de un registro.

3. Tablas

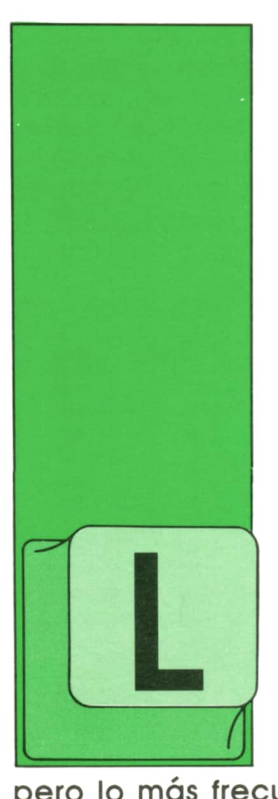
La organización de los datos en tablas implica su almacenamiento en posiciones consecutivas de memoria, de forma que se puede localizar cualquier elemento mediante la programación del algoritmo adecuado.

En caso de tablas o matrices de dos dimensiones, el almacenamiento debe ser lineal, es decir, por filas o columnas.

NOMBRE1	TFNO1
NOMBRE2	TFNO2
NOMBRE3	TFNO3
NOMBRE4	TFNO4
NOMBRE5	TFNO5
NOMBRE6	TFNO6

Ejemplo de matriz de dos dimensiones. Cada registro se almacena en una posición consecutiva, dentro del área reservada a la tabla en memoria.

MAQUINA 8088



LOS ficheros de texto están constituidos por caracteres legibles agrupados en líneas. Las líneas son de longitud variable y pueden llegar a tener hasta 253 caracteres, pero lo más frecuente es que no pasen de 40 u 80, que son los que caben en el ancho de la pantalla. El fin de cada línea está señalado internamente por dos caracteres de control: «retorno de carro» y «nueva línea». Estos dos caracteres tienen los códigos ASCII 13 y 10, respectivamente.

Los programas fuente de la mayoría de los lenguajes de programación son, desde el punto de vista físico, ficheros de texto estándares. Los del ensamblador se diferencian del estándar únicamente en que la longitud máxima en la línea está restringida a 132 caracteres.



Sentencias

Las unidades lógicas del lenguaje ensamblador del 8088 (MASM) son las sentencias. Las sentencias deben escribirse en líneas independientes y pueden estar constituidas por cuatro tipos de componentes que deben ir separados por al menos un espacio en blanco. Los componentes de las sentencias son:

1. Nombre (opcional).
2. Operación.
3. Conjunto de operandos.
4. Comentario (opcional).

Excepto para los textos que constituyen mensajes o datos, las letras mayúsculas y minúsculas son equivalentes a todos los efectos.

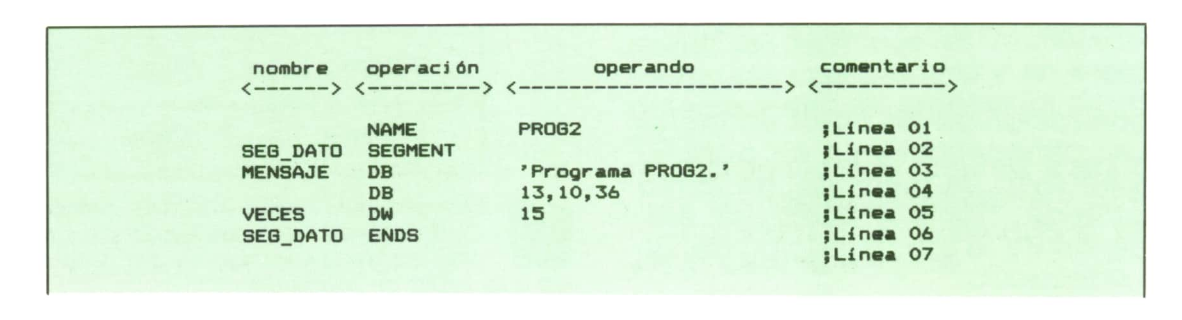


Ejemplo

Vamos a usar el siguiente programa fuente (que llamaremos PROG2) para comentar los componentes de los diferentes tipos de sentencias.

Cuando sometamos dicho programa a los procesos de ensamblaje y montaje (explicado en el tema anterior), veremos que en la ejecución del mismo se genera un mensaje repetidamente sobre la pantalla.

Los cuatro componentes de las sentencias se han organizado por columnas en el listado para hacer más fácil su identificación.



```
 nombre    operación          operando             comentario
<------>  <--------->  <-------------------->  <---------->

          NAME         PROG2                    ;Línea 01
SEG_DATO  SEGMENT                               ;Línea 02
MENSAJE   DB           'Programa PROG2.'        ;Línea 03
          DB           13,10,36                 ;Línea 04
VECES     DW           15                       ;Línea 05
SEG_DATO  ENDS                                  ;Línea 06
                                                ;Línea 07
```

```
SEG_CODE  SEGMENT                                   ;Línea 08
PROC1     PROC        FAR                           ;Línea 09
          ASSUME      CS:SEG_CODE,DS:SEG_DATO       ;Línea 10
                                                    ;Línea 11
          PUSH        DS                            ;Línea 12
          MOV         AX,0000                       ;Línea 13
          PUSH        AX                            ;Línea 14
                                                    ;Línea 15
          MOV         AX,DATOS                      ;Línea 16
          MOV         DS,AX                         ;Línea 17
                                                    ;Línea 18
          MOV         CX,VECES                      ;Línea 19
                                                    ;Línea 20
BUCLE:    LEA         DX,MENSAJE                    ;Línea 21
          MOV         AH,9                          ;Línea 22
          INT         21H                           ;Línea 23
          LOOP        BUCLE                         ;Línea 24
                                                    ;Línea 25
          RET                                       ;Línea 26
PROC1     ENDP                                      ;Línea 27
                                                    ;Línea 28
SEG_CODE  ENDS                                      ;Línea 29
          END         PROC1                         ;Línea 30

<------> <---------> <----------------------> <----------->
 nombre   operación         operando             comentario
```



Nombre

El primer componente de la sentencia es casi siempre opcional y consiste en un nombre o etiqueta que sirve para identificar instrucciones aisladas o grupos de instrucciones que deban ser referenciadas desde otros puntos del programa.

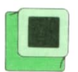
Los nombres deben cumplir las siguientes reglas:

1. Aunque se pueden especificar nombres de cualquier longitud, sólo los primeros 31 caracteres son reconocidos por el ensamblador.
2. Son válidos los siguientes caracteres:

— Las letras del alfabeto inglés (donde se excluye la ñ).
— Los números del 0 al 9.
— Los caracteres especiales de interrogación (?), punto (.), @ (ə), subrayado (_) y dólar ($).

3. Los números no pueden constituir el primer carácter de un nombre.
4. El punto (.) sólo puede usarse como primer carácter.

En el programa ejemplo PROG2, el nombre SEG_DATO que aparece en las líneas 2 y 6 especifica el nombre de un segmento de datos. El nombre VECES corresponde a una variable que se define en la línea 5 y se usa en la 19. Y BUCLE es una etiqueta que se le ha asignado a la instrucción ejecutable de la línea 21 para poderla referenciar desde la línea 24.



Operación

El segundo componente es el más importante, ya que es el que realmente define la esencia de la sentencia. Hay dos tipos de operaciones totalmente diferentes que podemos llamar:

— Operaciones propiamente dichas.
— Seudo-operaciones.

Las operaciones propiamente dichas son códigos nemotécnicos de las acciones que puede realizar el microprocesador y que el ensamblador se encargará de traducir a código máquina. A las sentencias que contienen operaciones de este tipo las podemos llamar «sentencias ejecutables».

En el programa anterior son sentencias ejecutables las líneas que van de la 12 a la 26, excepto las número 15, 18, 20 y 25, que por legibilidad se han dejado en blanco. En ellas aparecen las operaciones PUSH, MOV, LEA, INT, LOOP y RET.

Se llaman seudo-operaciones a los códigos de operación que no generan código ejecutable. Los cuales, a su vez, pueden ser de dos tipos:

— Seudo-operaciones de definición de datos.
— Seudo-operaciones directivas.

Las primeras, como su nombre indica, sirven para definir datos, es decir, asignar nombres a trozos de memoria y hacer que el ensamblador genere los códigos (no ejecutables) con los valores iniciales de las mismas. Las seudo-operaciones DB y DW que aparecen en las líneas 3, 4 y 5 del programa PROG2 son de este tipo.

Las segundas sirven para definir al ensamblador determinadas circunstancias que influirán en la traducción de otras operaciones, pero no generan directamente ningún código. En el ejemplo anterior, son seudo-operaciones directivas las siguientes: NAME, SEGMENT, ENDS, PROC, ASSUME, ENDP y END.



Conjunto de operandos

El tercer componente es una información complementaria de la operación o de la seudo-operación y está constituida por uno o más elementos. Cuando la sentencia contiene un código de operación estos elementos reciben el nombre de operandos. Con ellos se especifican los elementos involucrados que pueden ser: registros, referencias a la memoria, constantes o expresiones.

Cada operación tiene un número de operandos propio. Algunas no requieren ninguno, como es el caso de la operación RET en la línea 26. Otras requieren un operando, como ocurre con las operaciones PUSH, INT y LOOP. Finalmente, otras operaciones como MOV y LEA requieren dos operandos separados por una coma.



Comentario

El cuarto y último componente de las sentencias es siempre opcional y consiste en una cadena de caracteres iniciada por el carácter punto y coma. Los comentarios sirven para documentar el funcionamiento de los programas y cumplen una misión muy importante, ya que los códigos de operación de sólo 3 ó 4 letras, el profuso uso de los registros y la propia versatilidad del lenguaje hacen que los programas sin comentarios sean difícilmente comprensibles.

En nuestro ejemplo hemos usado el campo de comentario para numerar todas las líneas y poder referirnos a cada una de ellas sin confusión.



Estructura de los programas

El lenguaje ensamblador está diseñado para que se puedan escribir programas de muy diversa complejidad. Desde programas elementales de sólo 30 líneas, como el que acabamos de comentar, a aplicaciones muy complejas constituidas por decenas de miles de instrucciones.

Para facilitar la escritura y puesta a punto de los programas, el ensamblador permite dividir el código fuente en tantos módulos como se desee. Como se vio en el tema anterior, dichos módulos se ensamblan por separado generando programas objetos que posteriormente se reúnen como el LINK para producir el programa ejecutable.

También se ha dicho anteriormente que el microprocesador 8088 maneja cuatro áreas de memoria, cada una de las cuales se direcciona con dos registros que reciben el nombre de «registro de segmento» y «registro de desplazamiento».

Recordemos que con un determinado valor del «registro de segmento» sólo se puede abarcar 64 Kbytes de memoria y que si se quiere acceder fuera de esa área hay que modificar forzosamente el «registro de segmento».

Pues bien, debido a esta característica del 8088 es por lo que la memoria manejada por los programas debe estar particionada en bloques no mayores de 64 Kbytes, que en la terminología del ensamblador se denominan segmentos.

Esto significa que en todas las sentencias de los módulos fuente que generan código tienen que estar encuadradas en segmentos.

La división de un módulo en segmentos se puede elegir libremente con la única restricción de no pasar de los 64 Kbytes en cada uno de ellos. Pero hay que tener en cuenta que el programa debe incluir instrucciones específicas para di-

reccionar cada uno de los segmentos. Lo más usual en programas sencillos es formar un segmento con los datos y otro con las sentencias ejecutables, como en el ejemplo anterior. En dicho ejemplo las instrucciones específicas para direccionar el segmento de datos están en las líneas 16 y 17.



Las sentencias SEGMENT y ENDS

Los segmentos se definen con dos sentencias, la sentencia SEGMENT, que define el comienzo, y la sentencia ENDS, que indica el final. En medio de ellas se codifican todas las sentencias que constituyen el segmento.

El formato de estas sentencias es el siguiente:

```
nombre-seg SEGMENT [alineamiento] [tipo] ['clase']
            .
            .
            .
nombre-seg ENDS
```

En la figura anterior y en otras sucesivas que irán saliendo, vamos a utilizar la siguiente nomenclatura: lo escrito con minúsculas son nombres que deben ser elegidos por el usuario; lo escrito con mayúsculas son nombres que deben ser codificados como se indica; y lo encerrado entre corchetes son elementos opcionales que pueden ser omitidos.

nombre-seg. Es el nombre que se le quiere asignar al segmento y que debe aparecer, tanto en la sentencia SEGMENT que indica el comienzo del segmento, como en la sentencia ENDS que indica el final. La elección del nombre debe hacerse de acuerdo con las reglas anteriormente expuestas.

alineamiento. Es un parámetro opcional con el que se puede imponer ciertas condiciones a la dirección del primer byte del segmento. Efectivamente, especificando las palabras WORD, PARA o PAGE se consigue que la dirección del comienzo del segmento esté alineada a frontera de palabra, párrafo o página de memoria, respectivamente. Es decir, que la dirección inicial del segmento sea múltiplo de 2, 16 ó 256. También se puede especificar BYTE, que es lo mismo que decir que no se impone ninguna condición a la dirección de comienzo de segmento. Cuado se omite este parámetro se realiza el alineamiento a frontera de párrafo, igual que si se hubiera especificado PARA.

tipo. Este parámetro es opcional, y sirve para controlar la posición relativa en que se colocarán los distintos segmentos cuando se reúnan para formar el módulo ejecutable. Es por tanto una información que el ensamblador pasará al LINK, que es el programa que construye el módulo ejecutable. Se pueden especificar las palabras PUBLIC, COMMON, STACK, MEMORY y AT. En relación con este parámetro, rigen las siguientes reglas:

— A los segmentos tipo PUBLIC del mismo nombre se les asignan zonas de memorias contiguas.

— A los segmentos tipo COMMON del mismo nombre se les asigna una única zona de memoria sobre la que todos ellos quedan superpuestos.

— El segmento para el que se especifique tipo STACK será el que se usará como pila de ejecución del 8088. El funcionamiento de la pila se explicará con detalle más adelante.

— El segmento para el que se especifique tipo MEMORY será el último segmento que se sitúe en memoria.

— El segmento para el que se especifique AT seguido de una dirección se considerará como una plantilla que servirá para referenciar la zona de memoria especificada.

'clase'. Este parámetro, que también es opcional, consiste en un nombre expresado entre comillas que define la clase a la que pertenecerá el segmento. Esta es una facilidad adicional para especificar el orden en que se desea que los distintos segmentos formen el módulo ejecutable, ya que el LINK situará juntos a todos los segmentos de la misma clase.

A continuación se dan los esquemas de la división en segmentos de tres programas, que podrían servir de ejemplo tanto para formar módulos ejecutables independientes como para formar un módulo ejecutable único.

```
<-----programa A----->

        NAME PROGA
;--------------------
DATOS1  SEGMENT PAGE
        ...
        ...
        ...
DATOS1  ENDS
;---------------------
CODIGO1 SEGMENT PUBLIC
        ...
        ...
        ...
CODIGO1 ENDS
;-----------------
PILA    SEGMENT STACK
        ...
        ...
        ...
PILA    ENDS
;-----------------
        END

<-----programa A----->
```

```
<-----programa B----->

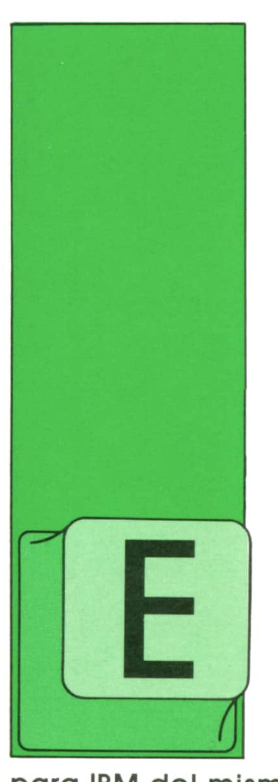
        NAME PROGB
;----------------------
DATOS2  SEGMENT MEMORY
        ...
        ...
        ...
DATOS2  ENDS
;----------------------
CODIGO1 SEGMENT PUBLIC
        ...
        ...
        ...
CODIGO1 ENDS
;-----------------
        END

<-----programa B----->
```

```
<----programa C---->

        NAME PROGC
;-----------------
CODIGO2 SEGMENT
        ...
        ...
        ...
CODIGO2 ENDS
;-----------------
        END

<----programa C---->
```



PROGRAMAS

EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS



Programas: quinielas

EL siguiente programa nos permitirá la realización de quinielas múltiples. El listado del programa 1 es únicamente para el SPECTRUM. El programa 2 es la versión para IBM del mismo.

```
  10    REM Mina two
  20   FLASH 1: PRINT AT 10,8;
       "PARE LA CINTA":
       PAUSE 500: FLASH 0: CLS
  25    DEF FN K(w,c)=(100*w)/c
  30   BEEP .08,20: BEEP .1,0:
        INPUT "Numero de partid
os a desarrollar >>>";nap
  35   DIM m(nap,2): DIM p$
      (nap,16): DIM M$(nap,3):
       DIM t(nap,2): DIM C$(nap
        250)
  40   LET tri=0: LET dob=0
  50   REM  Relleno de matrices
  80     FOR u=1 TO nap STEP 1
  90   BEEP .08,20: BEEP .1,0
 100  INPUT AT 0,0;("Partido ";
u);TAB 20;("Apuesta");AT 1,0;("=
========");TAB 20;("=======");AT
 2,0;p$(u);(".....>");TAB 20;M$(
u)
 110 REM Escritura en pantalla
 111   REM &&&&&&&&&&&&&&&&&&&&&&
&
 113   GO SUB 1000
 116  PRINT AT u+2,0;u;".-";p
      $(u): FOR v=1 TO 3:
       BEEP .04,30: PRINT AT
       u+2,19+v;M$(u,v): NEXT v
 117   NEXT u
 118    GO SUB 1000
 132   CLS
 137    GO SUB 3000
 140  PRINT AT 21,0;
      "Desea alguna correccion
? (s/n)"
 150   IF INKEY$="s" THEN
        PRINT AT 21,0;"
                         ":
         GO TO 180
 160   IF INKEY$="n" THEN
        PRINT AT 21,0;"
                         ": GO TO
        188
 170    GO TO 140
 180       PRINT AT 17,0;"Pulse l
a opcion adecuada";AT 19,2;"1.-
Nombres de equipos";AT 21,2;"2.-
 Apuestas (Signos)"
 183    IF INKEY$="1" THEN  PRIN
T AT 17,0;"
           ";AT 19,2;"
                      ";AT 21,2;"
                               ":
INPUT AT 0,0;("Numero de partido
 ?");t: BEEP .4,20: INPUT AT 0,0
;("Nueva opcion ?     ");p$(t):
GO TO 132
 186    IF INKEY$="2" THEN  PRIN
T AT 17,0;"
         ";AT 19,2;"
                  ";AT 21,2;"
                          ": INPUT
 AT 0,0;("Numero de partido ?");
t: BEEP .4,10: INPUT AT 0,0;("Nu
eva opcion ?     ");M$(t): GO TO
 132
 187    GO TO 180
 188    REM Formacion de t(nap,2
)
 189    FOR i=1 TO nap
 190    LET contsig=0
 191     FOR j=1 TO 3
 192   IF M$(i,j)<>" " THEN
        LET contsig=contsig+1
 193    NEXT j
 194   FOR j=1 TO 3
 196     IF M$(i,j)<>" " THEN  L
ET lug=j: LET j=3
 197    NEXT j
 198    LET t(i,1)=lug:
         LET t(i,2)=contsig
 199     NEXT i
 210    FOR l=1 TO nap
 220     LET m(l,2)=t(l,1)
 230    NEXT l
 250  REM #####Desarrollo#######
#
```

```
 299    LET k=1
 300   GO SUB 6000
 400    LET aux=nap
 410    LET i=aux
 420    IF m(i,2)<4 THEN
         GO SUB 5000: GO SUB 700
0:        GO TO 420
 430    LET m(i,2)=t(i,1)
 440    LET i=i-1
 450    IF i=0 THEN  CLS :
          GO SUB 7500: STOP
 460    LET m(i,2)=m(i,2)+1
 470    IF m(i,2)=4 THEN  GO TO
         480
 472    IF M$(i,m(i,2))=" " THEN
          GO TO 460
 474     GO TO 410
 480    LET m(i,2)=t(i,1): GO TO
        440
 999   REM .....sub.............
.
1000  PLOT 0,40: DRAW 252,0: DRA
W 0   ,116: DRAW -252,0: DRAW 0,
-116
1010 RETURN
1015    REM .....sub^^^^........
2999 REM ....sub...............
3000     FOR u=1 TO nap
3100    PRINT AT u+2,0;u;".-";p$
(        u): BEEP .02,-10: FOR v
         =1 TO 3: PRINT AT u+2,
         21+v;M$(u,v): NEXT v
3200    NEXT u
3250    GO SUB 1000
3500  RETURN
4000 REM ....sub^^^^...........
4999 REM ....sub........5000..
5000   FOR j=1 TO nap
5010   PRINT AT j+2,29;M$(j,m(j,
       2))
5011   LET C$(j,k)=M$(j,m(j,2))
5015    BEEP .009,20
5020     NEXT j
5021   PRINT AT 0,23;"VALIDA ?";
        AT 1,26;"(s/n)"
5022   IF INKEY$="n" THEN  GO TO
        5030
5023   IF INKEY$="s" THEN  LET
        K=K+1: GO TO 5030
5024    GO TO 5022
5025    PAUSE 0
5030    LET m(i,2)=m(i,2)+1
5035   IF m(i,2)>3 THEN  RETURN
5040    IF M$(i,m(i,2))=" " THEN
         GO TO 5030
```

```
5555  RETURN
5556 REM ....sub^^^^...........
6000  REM ...sub.costo..........
6005  LET tri=0: LET dob=0
6010  FOR p=1 TO nap
6020     IF t(p,2)=3 THEN  LET
           tri=tri+1: GO TO 6040
6030     IF t(p,2)=2 THEN  LET
           dob=dob+1
6040 NEXT p
6050     LET Numcol=INT ((2^dob)*
(3^tri))
6060     LET Costo=Numcol*15
6100  RETURN
6101 REM ..sub.^^^^^...........
7000 REM ....sub.porcen........
7030    PRINT AT 0,0;"COLUM.TOT.>
> ";INT (NUMCOL);AT 1,0;"COSTO T
OT.>> ";INT (COSTO);"Pts."
7040  PRINT AT 17,0;"Columnas se
leccionadas>> ";K-1;AT 21,0;"HAS
TA AHORA>> ";(K-1)*15;"Pts.";AT
19,1;(FN K(K-1,Numcol));TAB 12;"
% al14"
7100    RETURN
7101  REM ....sub^^^^...........
7499 REM ....sub..copiar.......
7500 LET h=0
7510 FOR x=0 TO k*2 STEP 2
7515     LET h=h+1
7520     FOR z=1 TO nap
7522   IF x>=30 THEN  PRINT AT 20
,1;"Quieres mas ? (s/n)": BEEP 1
,-20: GO TO 7560
7530     PRINT AT z,x;C$(z,h): BE
EP .01,20
7535    IF C$(z,h)=" " THEN  PRIN
T AT 20,0;"ESTO SE HA ACABADO.$$
 SUERTE $$": GO TO 7600
7545     GO TO 7580
7560    IF INKEY$="s" THEN  CLS :
 LET h=h-1: GO TO 7510
7565    IF INKEY$="n" THEN  CLS :
 PRINT AT 11,2;"BUENA SUERTE LES
 DESEA ESTE ZX-SPECTRUM": STOP
7570    GO TO 7560
7580  NEXT z
7585    NEXT x
7600    RETURN
7601 REM ....sub^^^^...........
```

```
1000 REM ***************************
1010 REM *    Q U I N I E L A S    *
1020 REM ***************************
1030 REM * POR Fco. Morales        *
1040 REM ***************************
1050 REM
1060 REM ***************************
1070 REM * (c) Ed. Siglo Cultural  *
1080 REM * (c) 1987                *
1090 REM ***************************
```

```
1100 REM
1110 DEF FNK(W,C)=(100*W)/C
1120 CLS
1130 LOCATE 10,1
1140 INPUT "Numero de partidos a desarrollar > ",NP
1150 DIM M(NP,2):DIM P$(NP):DIM M$(NP):DIM T(NP,2):DIM C$(NP)
1160 FOR I=1 TO NP
1170    LET C$(I)="     "
1180 NEXT I
1190 LET LR=0
1200 LET DO=0
1210 REM
1220 REM ***  RELLENO DE MATRICES  ***
1230 REM
1240 CLS
1250 PRINT "PARTIDO                                 APUESTA"
1260 PRINT "=======                                 ======="
1270 FOR U=1 TO NP
1280    LOCATE 3+U,1
1290    LINE INPUT "",P$(U)
1300    LOCATE 3+U,36
1310    LET SW=0
1320    LINE INPUT "",M$(U)
1330    IF LEN(M$(U))>3 THEN GOTO 1300
1340    FOR I=1 TO LEN(M$(U))
1350       IF MID$(M$(U),I,1)<>"1" AND MID$(M$(U),I,1)<>"X" AND MID$(M$(U),I,1)<
>"2" THEN LET SW=1
1360    NEXT I
1370    IF SW=1 THEN GOTO 1300
1380    LET M$(U)=LEFT$(M$(U)+"   ",3)
1390 NEXT U
1400 CLS:PRINT:PRINT:PRINT
1410 PRINT "No.  PARTIDO                            APUESTA"
1420 PRINT "===  =======                            ======="
1430 FOR U=1 TO NP
1440    LOCATE 6+U,1
1450    PRINT U
1460    LOCATE 6+U,6
1470    PRINT P$(U)
1480    LOCATE 6+U,33
1490    PRINT M$(U)
1500 NEXT U
1510 LOCATE 22,1
1520 PRINT "QUIERES MODIFICAR ALGUNO (S/N)
1530 LET A$=INKEY$:IF A$="" THEN GOTO 1530
1540 IF A$="S" OR A$="s" THEN 1570
1550 IF A$="N" OR A$="n" THEN GOTO 1720
1560 GOTO 1530
1570 LOCATE 22,1
1580 PRINT "1.- MODIFICAR PARTIDO              "
1590 PRINT "2.- MODIFICAR APUESTA              "
1600 LET A$=INKEY$:IF A$="" THEN GOTO 1600
1610 IF A$<>"1" AND A$<>"2" THEN GOTO 1600
1620 LOCATE 22,1
1630 PRINT SPACE$(80)
1640 LOCATE 22,1
1650 INPUT "DE QUE PARTIDO? ",U
1660 IF U<1 OR U>NP THEN GOTO 1620
1670 ON VAL(A$) GOTO 1680,1700
1680 INPUT "NOMBRE DEL PARTIDO > ",P$(U)
1690 GOTO 1400
1700 INPUT "APUESTA > ",M$(U)
1710 GOTO 1400
1720 LOCATE 22,1
1730 PRINT SPACE$(40)
1740 LOCATE 22,1
1750 PRINT "ESPERA UN MOMENTO"
1760 REM
```

```
1770 REM *** CALCULO ***
1780 REM
1790 FOR I=1 TO NP
1800    LET CT=0
1810    FOR J=1 TO 3
1820       IF MID$(M$(I),J,1)<>" " THEN LET CT=CT+1
1830    NEXT J
1840    FOR J=1 TO 3
1850       IF MID$(M$(I),J,1)<>" " THEN LET LG=J:LFT J=3
1860    NEXT J
1870    LET T(I,1)=LG
1880    LET T(I,2)=CT
1890 NEXT I
1900 FOR L=1 TO NP
1910    LET M(L,2)=T(L,1)
1920 NEXT L
1930 LOCATE 22,1
1940 PRINT SPACE$(40)
1950 REM
1960 REM *** DESARROLLO ***
1970 REM
1980 LET K=1
1990 GOSUB 2270
2000 LET AX=NP
2010 LET I=AX
2020 IF M(I,2)<4 THEN GOSUB 2110:GOSUB 2370:GOTO 2020
2030 LET M(I,2)=T(I,1)
2040 LET I=I-1
2050 IF I=0 THEN CLS:GOSUB 2470:END
2060 LET M(I,2)=M(I,2)+1
2070 IF M(I,2)=4 THEN GOTO 2100
2080 IF MID$(M$(I),M(I,2),1)=" " THEN GOTO 2060
2090 GOTO 2010
2100 LET M(I,2)=T(I,1):GOTO 2040
2110 REM
2120 FOR J=1 TO NP
2130    LOCATE 6+J,38
2140    PRINT MID$(M$(J),M(J,2),1)
2150    LET C$(J)=LEFT$(C$(J),K-1)+MID$(M$(J),M(J,2),1)+MID$(C$(J),K+1)
2160 NEXT J
2170 LOCATE 1,30
2180 PRINT "(VALIDA? (S/N)":BEEP
2190 LET A$=INKEY$
2200 IF A$="N" OR A$="n" THEN GOTO 2230
2210 IF A$="S" OR A$="s" THEN LET K=K+1:GOTO 2230
2220 GOTO 2190
2230 LET M(I,2)=M(I,2)+1
2240 IF M(I,2)>3 THEN RETURN
2250 IF MID$(M$(I),M(I,2),1)=" " THEN GOTO 2230
2260 RETURN
2270 REM
2280 LET TI=0
2290 LET DB=0
2300 FOR P=1 TO NP
2310    IF T(P,2)=3 THEN LET TI=TI+1:GOTO 2330
2320    IF T(P,2)=2 THEN LET DB=DB+1
2330 NEXT P
2340 LET NM=INT((2^DB)*(3^TI))
2350 LET CS=NM*15
2360 RETURN
2370 REM
2380 LOCATE 1,1
2390 PRINT "COLUMNAS TOTALES = ";INT(NM)
2400 LOCATE 2,1
2410 PRINT "COSTO TOTAL       = ";INT(CS);" PTS."
2420 LOCATE 22,1
2430 PRINT "COLUMNAS SELECCIONADAS = ";K-1
2440 PRINT "HASTA AHORA ";(K-1)*15;" PTS."
```

```
2450 PRINT FNK(K-1,NM);" % A LOS 14";
2460 RETURN
2470 CLS
2480 LET H=0
2490 FOR X=0 TO K*2 STEP 2
2500    LET H=H+1
2510    FOR Z=1 TO NP
2520       IF X>=40 THEN LOCATE 22,1:PRINT "QUIERES MAS (S/N)":GOTO 2570
2530       LOCATE Z+1,X+1
2540       PRINT MID$(C$(Z),H,1)
2550       IF MID$(C$(Z),H,1)=" " THEN LOCATE 22,1:PRINT "ESTO SE HA ACABADO":PR
INT "SUERTE":RETURN
2560       GOTO 2610
2570       LET A$=INKEY$
2580       IF A$="S" OR A$="s" THEN CLS:LET H=H-1:GOTO 2490
2590       IF A$="N" OR A$="n" THEN LOCATE 22,1:PRINT "BUENA SUERTE":END
2600       GOTO 2570
2610    NEXT Z
2620 NEXT X
2630 RETURN
```

Las correcciones que hay que hacer para que el programa pueda funcionar en el COMMODORE, AMSTRAD y MSX son las siguientes:

COMMODORE:

```
1120 PRINT CHR$(147)
1130 POKE 214,9:POKE 211,0
1240 PRINT CHR$(147)
1280 POKE 214,2+U:POKE 211,0
1300 POKE 214,2+U:POKE 211,35
1400 PRINT CHR$(147):PRINT:PRINT:PRINT
1440 POKE 214,5+U:POKE 211,0
1460 POKE 214,5+U:POKE 211,5
1480 POKE 214,5+U:POKE 211,32
1510 POKE 214,21:POKE 211,0
1530 GET A$
1570 POKE 214,21:POKE 211,0
1660 GET A$
1620 POKE 214,21:POKE 211,0
1630 FOR I=1 TO 80:PRINT " ";:NEXT I
1640 POKE 214,21:POKE 211,0
1720 POKE 214,21:POKE 211,0
1730 FOR I=1 TO 80:PRINT " ";:NEXT I
1740 POKE 214,21:POKE 211,0
1930 POKE 214,21:POKE 211,0
1940 FOR I=1 TO 40:PRINT " ";:NEXT I
2130 POKE 214,5+J:POKE 211,37
2170 POKE 214,0:POKE 211,29
2190 GET A$
2380 POKE 214,0:POKE 211,0
2400 POKE 214,21:POKE 211,0
2470 PRINT CHR$(147)
2520 IF X>=39 THEN POKE 214,21:POKE 211,0:PRINT "QUIERES MAS (S/N)":GOTO 2570
2530 POKE 214,Z:POKE 211,X
2550 IF MID$(C$(Z) ,H,1)=" " THEN POKE 214,21:POKE 211,0:PRINT "ESTO SE HA ACABADO":PRINT "SUERTE":RETURN
2570 GET A$
2580 IF A$="S" OR A$="s" THEN PRINT CHR$(147):LET H=H+1:GOTO 2490
2590 IF A$="N" OR A$="n" THEN POKE 214,21:POKE 211,0:PRINT "BUENA SUERTE":END
```

AMSTRAD Y MSX:

Para que el programa funcione en estos ordenadores lo único que hay que hacer es cambiar de orden los argumentos de todas las sentencias LOCATE del programa. Por ejemplo, si tenemos la línea 1440

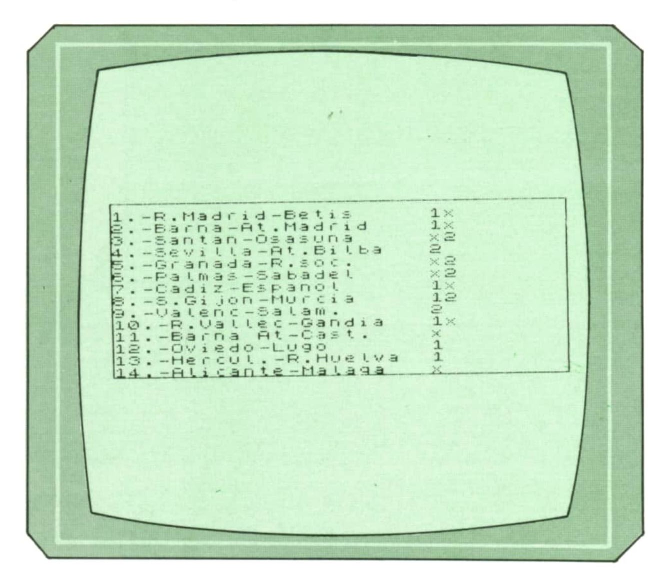
```
1440 LOCATE 6+U,1
```

para pasar esta línea al MSX y al AMSTRAD tendremos que poner:

```
1440 LOCATE 1,6+U
```

Este programa nos permitirá meter el nombre de hasta veinticuatro equipos de fútbol emparejados de dos en dos. Una vez que éstos se encuentran en memoria, el programa nos pedirá que rellenemos la quiniela como nosotros queramos. A

continuación nos aparecerán en la pantalla la lista de todos los equipos que hemos introducido seguida de la lista de los resultados de los partidos. Se le preguntará al usuario si quiere realizar alguna modificación. En caso afirmativo, se preguntará el número de partido, si queremos cambiar el nombre o el resultado y la variación que queremos realizar.





¿Desea alguna corrección? (s/n).

Si no queremos realizar ninguna modificación o ya hemos modificado todo lo que queríamos, nos aparecerá una nueva pantalla dentro de la cual se desarrollará todo el resto del programa. En esta pantalla, aparte de aparecer la lista de los partidos y las apuestas, nos aparecerán los siguientes datos:

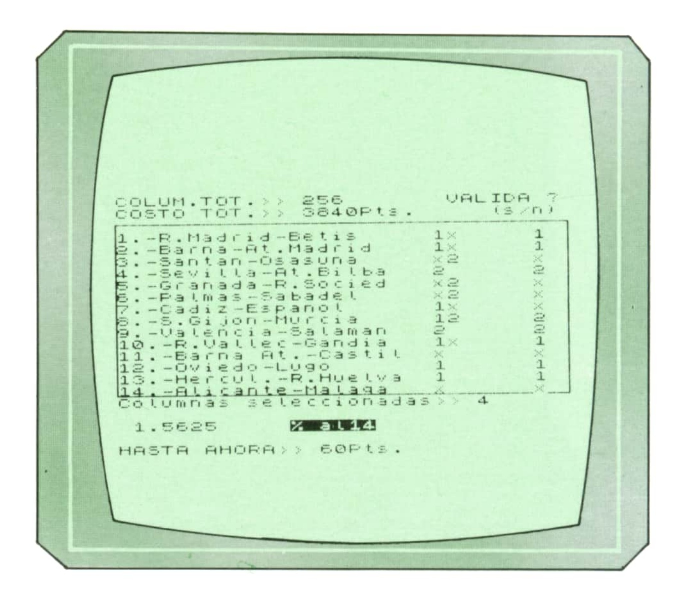
— Número total de columnas realizadas.

— Coste total de la apuesta con todas las columnas.

— Posibilidades de sacar catorce resultados.

— Número de columnas seleccionadas.

— Coste de las columnas seleccionadas.



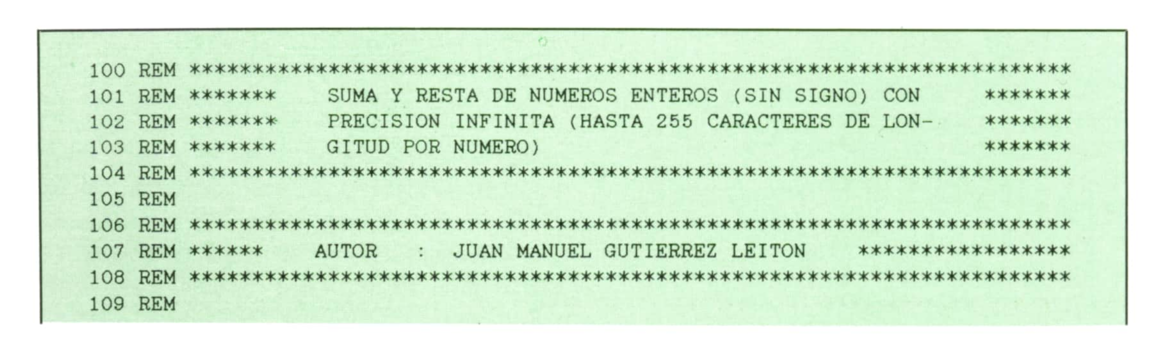
A continuación irán desfilando ante nuestros ojos todas las columnas posibles en la quiniela. Cada vez que aparezca una columna tendremos que pulsar si es válida o no. En caso negativo, el ordenador no la tendrá en cuenta, pero en caso positivo, la almacenará para volver a imprimirla más tarde.

Una vez que se han visualizado todas las columnas posibles, éstas aparecerán por pantalla de forma independiente y una a una.



Programa: Cálculos infinitos

El siguiente programa que vamos a ver nos permitirá sumar y restar números enteros de cualquier longitud. El programa está preparado para detectar errores. Por ejemplo, si en vez de un número tecleamos una letra o un signo de puntuación, el ordenador nos dirá que nos hemos equivocado.



```
100 REM ***********************************************************************
101 REM *******    SUMA Y RESTA DE NUMEROS ENTEROS (SIN SIGNO) CON      *******
102 REM *******    PRECISION INFINITA (HASTA 255 CARACTERES DE LON-     *******
103 REM *******    GITUD POR NUMERO)                                    *******
104 REM ***********************************************************************
105 REM
106 REM ***********************************************************************
107 REM ******     AUTOR   :  JUAN MANUEL GUTIERREZ LEITON     *****************
108 REM ***********************************************************************
109 REM
```

```
110 REM ***********************************************************************
111 REM ***********    (C). EDICIONES SIGLO CULTURAL, 1987    *****************
112 REM ***********************************************************************
113 CLS
114 INPUT "Dame el primer numero ";N$(1)
115 LET R$=N$(1)
116 GOSUB 164
117 IF ER=1 THEN GOSUB 174 :GOTO 113
118 LET L1=LEN(N$(1))
119 LET T=1
120 GOSUB 316
121 LET L1=L1-NC
122 INPUT "Dame el segundo numero";N$(2)
123 LET R$=N$(2)
124 GOSUB 164
125 IF ER=1 THEN GOSUB 174 :GOTO 122
126 LET L2=LEN(N$(2))
127 LET T=2
128 GOSUB 316
129 LET L2=L2-NC
130 INPUT "dame el tipo de operador (+,-)";O$
131 IF LEN(O$)>1 THEN PRINT"OPERACION NO VALIDA":GOTO 130
132 IF O$<>"+" AND O$<>"-" THEN PRINT"OPERACION NO VALIDA":GOTO 130
133 IF O$="+" THEN GOTO 156
134 LET F=1
135 IF L1>L2 THEN GOTO 137
136 GOTO 139
137 GOSUB 188
138 GOTO 160
139 IF L1<L2 THEN GOTO 141
140 GOTO 149
141 LET L=L1
142 LET L1=L2
143 LET L2=L
144 LET I$=N$(1)
145 LET N$(1)=N$(2)
146 LET N$(2)=I$
147 GOSUB 188
148 GOTO 159
149 IF N$(1)<N$(2) THEN 151
150 GOTO 156
151 LET I$=N$(1)
152 LET N$(1)=N$(2)
153 LET N$(2)=I$
154 GOSUB 255
155 GOTO 159
156 IF L1>L2 THEN GOSUB 188:GOTO 160
157 IF L1<L2 THEN GOSUB 215:GOTO 160
158 GOSUb 255 :GOTO 160
159 A$=A$+"-"
160 GOSUB 285
161 PRINT N$(3)
162 END
163 REM
164 REM ***********************************************
165 REM ****** RUTINA DE CHEQUEO DE LOS NUMEROS ******
166 REM ***********************************************
167 REM
168 FOR I=1 TO LEN(R$)
169    K$=MID$(R$,I,1)
170    IF K$<"0" OR K$>"9" THEN LET ER=1:GOTO 173
171 NEXT I
172 LET ER=0
173 RETURN
174 REM **********************************************************************
175 REM ************ rutina de errores de entrada ****************************
176 REM **********************************************************************
```

```
177 CLS
178 PRINT "EL CARACTER ";
179 PRINT K$;
180 PRINT "  NO ES VALIDO"
181 PRINT:PRINT:PRINT:PRINT:PRINT
182 PRINT "PULSE UNA TECLA PARA VOLVER A EMPEZAR"
183 LET A$=INKEY$
184 IF A$="" THEN GOTO 183
185 CLS
186 RETURN
187 REM
188 REM *******************************************************************
189 REM * SUBRUTINA DE SUMA/RESTA DE STRINGS CUANDO EL N1 ES MAYOR QUE N2 *
190 REM *******************************************************************
191 REM
192 LET AC=0
193 LET A$=""
194 LET J=L1
195 FOR I=L2 TO 1 STEP -1
196     LET C$=MID$(N$(1),J,1)
197     LET D$=MID$(N$(2),I,1)
198     LET J=J-1
199     IF F=1 THEN GOSUB 299 :GOTO 201
200     GOSUB 240
201 NEXT I
202 FOR I=L1-L2 TO 1 STEP -1
203     LET C$=MID$(N$(1),I,1)
204     LET D$="0"
205     IF F=1 THEN GOSUB 299 :GOTO 207
206     GOSUB 240
207 NEXT I
208 IF F<>1 THEN GOSUB 273
209 RETURN
210 REM
211 REM ************************************************
212 REM * FIN DE LA SUBRUTINA DE SUMA/RESTA DE STRINGS *
213 REM ************************************************
214 REM
215 REM **********************************************************
216 REM * SUBRUTINA DE SUMA DE STRINGS CUANDO N2 ES MAYOR QUE N1 *
217 REM **********************************************************
218 REM
219 LET AC=0
220 LET A$=""
221 LET J=L2
222 FOR I=L1 TO 1 STEP -1
223     LET C$=MID$(N$(1),I,1)
224     LET D$=MID$(N$(2),J,1)
225     LET J=J-1
226     GOSUB 240
227 NEXT I
228 FOR I=L2-L1 TO 1 STEP -1
229     LET C$="0"
230     LET D$=MID$(N$(2),I,1)
231     GOSUB 240
232 NEXT I
233 GOSUB 273
234 RETURN
235 REM
236 REM ********************************************************************
237 REM * FIN DE LA SUBRUTINA DE SUMA DE STRINGS CUANDO N2 ES MAYOR QUE N1 *
238 REM ********************************************************************
239 REM
240 REM ***************************************
241 REM * SUBRUTINA DE SUMA DE DOS CARACTERES *
242 REM ***************************************
243 REM
244 LET SU=VAL(C$)+VAL(D$)+AC
```

```
245 IF SU >= 10 THEN LET AC=1
246 IF SU < 10  THEN LET AC=0
247 N$=CHR$((SU MOD 10)+ASC("0"))
248 LET A$=A$+N$
249 RETURN
250 REM
251 REM *************************************************
252 REM * FIN DE LA SUBRUTINA DE SUMA DE DOS CARACTERES *
253 REM *************************************************
254 REM
255 REM **************************************************************
256 REM * SUBRUTINA DE SUMA/RESTA DE STRINGS CUANDO N1 ES IGUAL A N2 *
257 REM **************************************************************
258 REM
259 LET AC=0
260 LET A$=""
261 FOR I=L1 TO 1 STEP -1
262    LET C$=MID$(N$(1),I,1)
263    LET D$=MID$(N$(2),I,1)
264    IF F=1 THEN GOSUB 299 :GOTO 266
265    GOSUB 240
266 NEXT I
267 IF F<>1 THEN GOSUB 273
268 RETURN
269 REM
270 REM ************************************************
271 REM * FIN DE LA SUBRUTINA DE SUMA/RESTA DE STRINGS *
272 REM ************************************************
273 REM
274 REM ***************************************************
275 REM * SUBRUTINA POR SI LA ULTIMA SUMA DA DE ACARREO 1 *
276 REM ***************************************************
277 REM
278 IF AC=1 THEN LET A$=A$+"1"
279 RETURN
280 REM
281 REM **********************************
282 REM * FIN DE LA SUBRUTINA DE ACARREO *
283 REM **********************************
284 REM
285 REM **********************************
286 REM * SUBRUTINA DE INTERCAMBIO DE A$ *
287 REM **********************************
288 REM
289 LET N$(3)=""
290 FOR I=LEN(A$) TO 1 STEP -1
291    LET N$(3)=N$(3)+MID$(A$,I,1)
292 NEXT I
293 RETURN
294 REM
295 REM ********************************************
296 REM * FIN DE LA SUBRUTINA DE INTERCAMBIO DE A$ *
297 REM ********************************************
298 REM
299 REM ****************************************
300 REM * SUBRUTINA DE RESTA DE DOS CARACTERES *
301 REM ****************************************
302 REM
303 IF VAL(C$)<(VAL(D$)+AC) THEN GOTO 307
304 LET RE=VAL(C$)-(VAL(D$)+AC)
305 LET AC=0
306 GOTO 309
307 LET RE=10+(VAL(C$)-(VAL(D$)+AC))
308 LET AC=1
309 LET N$=CHR$(RE+ASC("0"))
310 LET A$=A$+N$
311 RETURN
312 REM
```

```
313 REM ***************************************************
314 REM * FIN DE LA SUBRUTINA DE RESTA DE DOS CARACTERES *
315 REM ***************************************************
316 REM
317 REM ************************************************************
318 REM * SUBRUTINA DE SANGRADO DE N$(i) POR SI EMPIEZA POR CEROS *
319 REM ************************************************************
320 REM
321 LET NC=0
322 FOR I=1 TO LEN (N$(T))
323    IF MID$(N$(T),I,1)="0" THEN LET NC=NC+1:GOTO 325
324    GOTO 326
325 NEXT I
326 IF NC<>0 THEN GOTO 328
327 GOTO 333
328 LET R$=N$(T)
329 LET N$(T)=""
330 FOR I=NC+1 TO LEN(R$)
331    LET N$(T)=N$(T)+MID$(R$,I,1)
332 NEXT I
333 RETURN
334 REM
335 REM *********************************************************************
336 REM * FIN DE LA SUBRUTINA DE SANGRADO POR SI N$(i) EMPIEZA POR CEROS *
337 REM *********************************************************************
```

El programa es válido para todos los ordenadores con sólo modificar algunas líneas. El programa funcionará perfectamente y sin cambios en el IBM, AMSTRAD y MSX. Para el SPECTRUM y COMMODORE, estas modificaciones son:

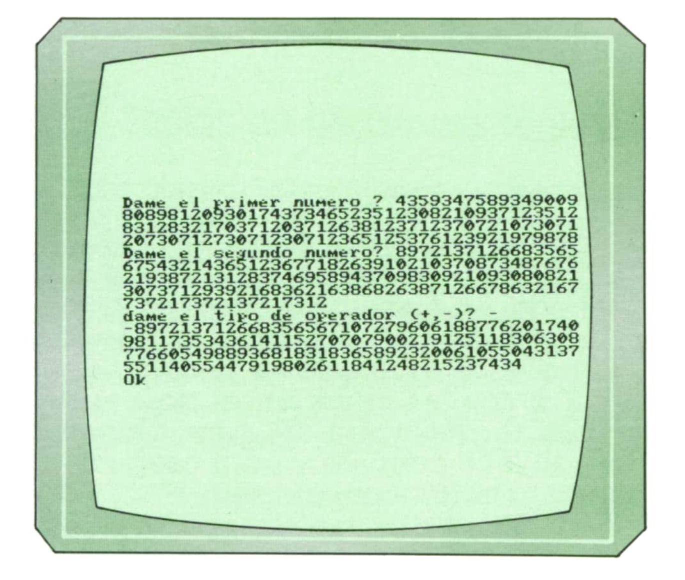
COMMODORE:

```
113 PRINT CHR$(147)
177 PRINT CHR$(147)
183 GET A$
185 PRINT CHR$(147)
247 LET N$=CHR$ ((SU-INT(SU/10))+ASC("0"))
N$=CHR$((SU–INT(SU/10)0+ASC("0")
```

SPECTRUM:

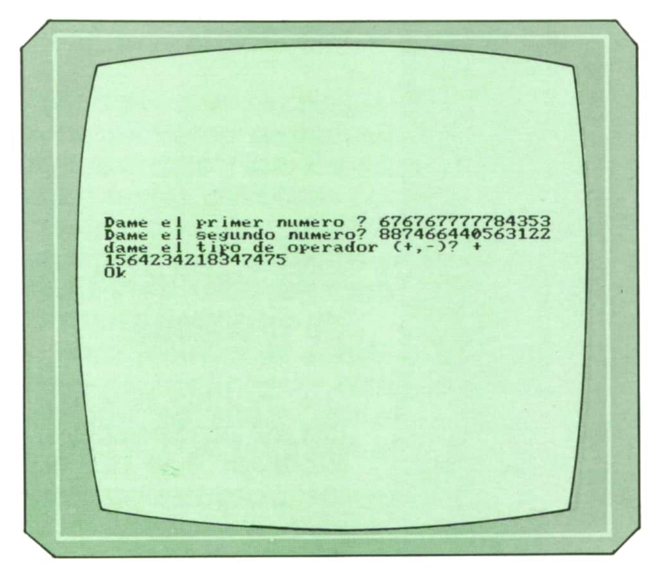
```
113 CLS:DIM N$(3,300)
159 LET A$=A$+"–"
162 STOP
169 LET K$=R$(I)
196 LET C$=N$(1,J)
197 LET D$=N$(2,I)
203 LET C$=N$(1,I)
223 LET C$=N$(1,I)
224 LET D$=N$(2,J)
230 LET D$=N$(2,I)
247 LET N$=CHR$((SU–INT(SU/10))+CO-
DE("0"))
262 LET C$=N$(1,I)
263 LET D$=N$(2,1)
```

```
291 LET N$(3)=N$(3)+A$(I)
309 LET N$=CHR$(RE+CODE("0"))
323 IF N$(T,I)="0" THEN LET NC=NC+1:GO-
TO 325
331 LET N$(T)=N$(T)+R$(I)
```





Resultado de restar dos números muy largos.





Resultado de sumar dos números muy cortos.

TECNICAS DE ANALISIS

ESTRUCTURA Y PROCESOS INTERNOS

Definición de procesos



ON mucha frecuencia los procesos a establecer con los datos, para su presentación en los informes de salida, son sencillos e, incluso en ocasiones, obvios; en estos casos, una breve anotación para el programador puede ser suficiente. Por el contrario, en numerosos casos es útil hacer una definición precisa y expresa de los procesos y cálculos a realizar. Conviene entonces reunir en el dossier del análisis la definición de estos procesos para su fácil consulta y para asegurar la coherencia total del proceso.

Normalmente se suele preparar un documento preimpreso (como el que aparece en la figura 1) para una más clara y concisa presentación de los datos.

DEFINICION DE PROCESO

Nombre ....

Código ....

Programa ....

UTILIDAD ....

....

....

Definición	Operandos

OPERANDOS

Nombre	Descripción

Se comienza con los datos de identificación: nombre del proceso (si se quiere dar alguno), código de letras y número (normalmente referido a la aplicación y al programa de que se trata) y referencia del programa en que se realizarán los cálculos; se suele incluir un comentario, a continuación, sobre el objeto general del proceso que se describe. El cuerpo del documento está formado por la definición de los cálculos a realizar (descripción liteal, fórmulas, etc.) con indicación, a la derecha y para cada cálculo descrito, de los operandos que intervienen.

En esta primera parte se suele poner el nombre mnemotécnico de los operandos para, en la parte inferior del documento, describirlos con detalle, haciendo referencia al resto de documentos en los que estos datos de entrada vienen especificados.



Tablas de decisiones lógicas

Para aclarar las relaciones entre los datos a procesar se suele incluir un documento adicional donde se indiquen expresamente estas relaciones. Puede ser una simple lista de relaciones, en el caso más sencillo, o bien una tabla de decisión como las ya descritas anteriormente en estas notas. Es importante que las relaciones queden patentes de un modo claro: el analista debe decidir el modo de presentación de la información.



Datos interrelacionados e históricos

Es básico en cualquier aplicación el establecimiento de ficheros históricos para recuperación de los datos en el futuro, bien en los procesos normales (cierre de ciertos períodos de funcionamiento del sistema), bien cuando se produce algún error y es necesario rehacer los ficheros básicos del proceso.

Para reseñar el establecimiento y manejo de los ficheros históricos de la aplicación se suele utilizar un formulario como el que aparece en la figura 2.

FICHERO DE DATOS HISTORICOS

Nombre del fichero .... Código ....

Versiones existentes		
Identificación	Contenido	Vigencia

Paso	Proceso de entrada	Proceso de salida	Versión

Campo	Contenido	Situación	Vigencia

En él aparece, junto al nombre e identificación (código) del fichero, la identificación individual de las diferentes versiones con indicación de su contenido y expresión de la vigencia o utilidad en el tiempo.

En dos grupos de datos se suelen incluir: en uno de ellos, el manejo de las diferentes versiones existentes del fichero; en el otro, los campos que contienen. Se reseña para cada paso el proceso de entrada, el de salida (por sus códigos de identificación) y la versión del fichero que se utiliza. En el grupo de datos referido a la descripción de campos se han de señalar exclusivamente aquellos que son de interés desde el punto de vista de variación, con las diferentes versiones históricas existentes; para cada uno de ellos se indica su vigencia o caducidad, después de los datos necesarios para conocer su identificación y ubicación.

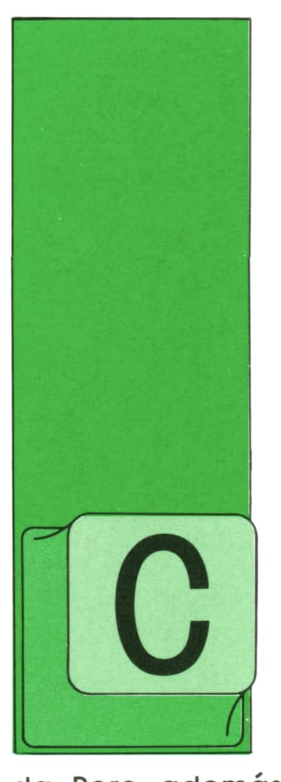


Relación general de campos, referencias y datos utilizados

Es útil incluir una relación general de todos los códigos que aparecen en la aplicación, con indicación de los datos necesarios para su descripción. Suele ser una lista donde, detrás de cada código, aparecen en dos columnas los documentos en los que se definen los códigos y los programas donde se utilizan. Es usual preparar dos relaciones de este tipo: una general, por riguroso orden alfabético de todos los códigos; otra en que aparecen agrupados (por tipo) los diferentes códigos existentes: campos de los ficheros, nombres de los propios ficheros, nombres de datos, nombres de programas, etc.

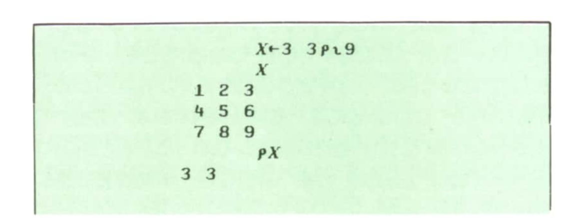
TECNICAS DE PROGRAMACION

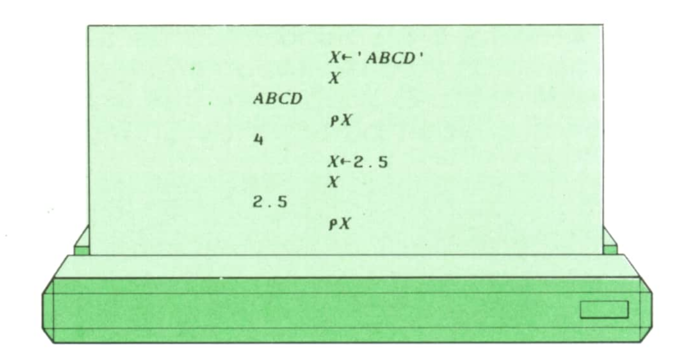
CONVERSION DE TIPO EN LA ASIGNACION DE VALORES A UNA VARIABLE



COMO hemos visto en los capítulos anteriores, una variable tiene un nombre y uno o varios valores, que pueden disponerse de acuerdo con una estructura determinada. Pero, además, estos valores pueden pertenecer a uno u otro tipo de datos: podrían ser numéricos o literales, enteros, reales en precisión normal o en precisión doble, etc. Uno de los problemas principales que hay que abordar en casi todos los lenguajes de programación consiste en saber qué sucede cuando a una variable de un tipo determinado se le asigna un valor de un tipo diferente. Por ejemplo, cuando a una variable de tipo entero se le asigna un valor con decimales, o viceversa.

Cada lenguaje de programación toma sus propias decisiones respecto a esto. En APL, por ejemplo, una variable no tiene tipo intrínseco, sino que toma automáticamente el tipo de los valores que se le asignan. La misma variable X podría ser, dependiendo de lo que se le asigne, ahora una matriz de números enteros, luego un vector de caracteres, más tarde un número único con decimales. Por lo tanto, la cuestión de la conversión de tipos no se presenta en este lenguaje. Veamos un ejemplo:

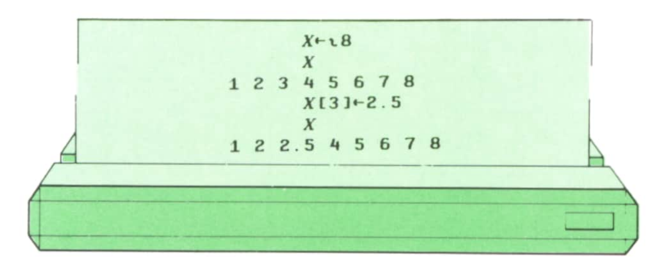




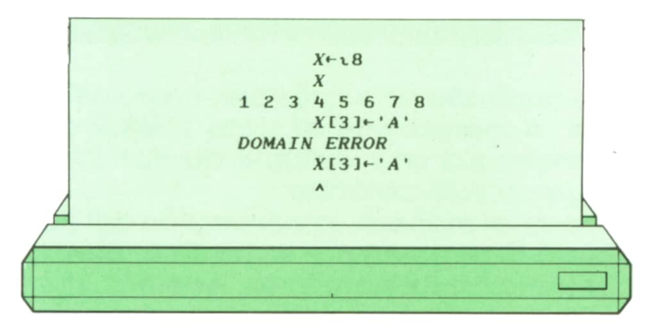
Se observará que, en APL, para preguntar cuál es el valor de una variable basta con escribir su nombre. Además, también es posible preguntar cuál es la estructura o forma de la variable, para lo cual basta con anteponer a su nombre la letra griega «rho». En el ejemplo anterior, primero hemos asignado a X una tabla de 3 filas y 3 columnas con valores iguales a los números del 1 al 9. Por eso, «rho» X nos salió igual a 3 3 (3 filas y 3 columnas). A continuación, asignamos a X la cadena de caracteres 'ABCD', lo que pudo realizarse sin ninguna dificultad, con lo que «rho» X pasó a ser 4 (pues 'ABCD' tiene cuatro elementos). Por último, asignamos a X el valor 2.5, un escalar, con lo que «rho» X vino a valer un vector vacío (ésta es, por convenio, la estructura de un escalar en APL), lo que se indica en el programa con una línea en blanco.

La cuestión es algo diferente cuando se intenta asignar un valor de cierto tipo a un solo elemento de una variable cuyo tipo es diferente. Por ejemplo, supongamos que tenemos una serie de datos formados por los números enteros del 1 al 8 y que le asignamos al tercer elemento de

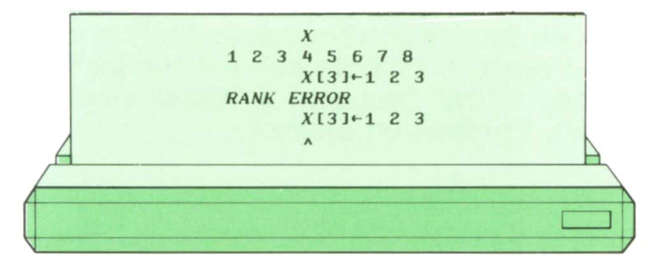
la serie el valor 2.5. Pues bien: en APL (donde externamente no hay más tipos que el numérico y el literal) la serie entera es convertida internamente, de forma automática, al tipo necesario para poder albergar un elemento con decimales:



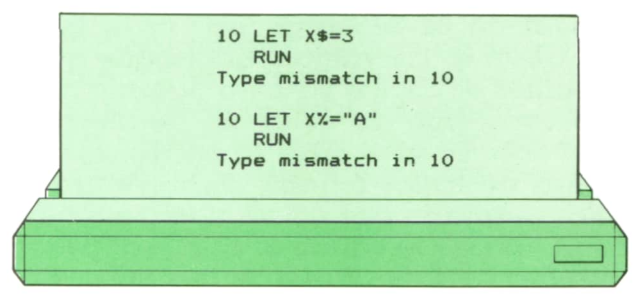
Es decir, a cualquier elemento de una serie o tabla numérica puede asignársele siempre un valor numérico cualquiera. Sin embargo, no debe asignársele un valor literal, pues números y caracteres no pueden mezclarse. Si lo intentamos, obtendremos un mensaje de error:



Tampoco será correcto tratar de asignar varios elementos a uno solo:



En BASIC, las variables tienen tipo. Esto quiere decir que una variable definida como entera, no podrá contener jamás, en ese programa, valores con decimales, y lo mismo ocurrirá con las variables literales, que no podrán recibir valores numéricos. Sin embargo, algunas asignaciones de valores de un tipo a variables de tipo diferente son permitidas por los intérpretes y compiladores BASIC, mientras que otras no lo son. No se permite nunca, por ejemplo, asignar un valor literal a una variable numérica de cualquier tipo y viceversa, como en los ejemplos siguientes:



Sin embargo, siempre es posible asignar un valor numérico de un tipo (entero, real de precisión normal o real de precisión doble) a una variable de un tipo diferente, aunque también numérico. Si el tipo de la variable es más amplio que el del valor asignado, la conversión se realiza automáticamente. Por ejemplo, el tipo real es siempre más amplio que el entero, pues lo comprende. El valor 3 (entero) puede considerarse también como 3.000000 (real de precisión normal) o como 3.00000000000000 (real de precisión doble). Asimismo, el tipo real de precisión doble es más amplio que el real de precisión simple: 2.567892 equivale a 2.56789200000000.

Pero también puede suceder que asignemos a una variable un valor de un tipo más amplio que el de aquélla. ¿Qué sucede entonces? Pues que el traductor de BASIC intenta convertir el valor asignado al tipo de la variable. A veces esto es posible sin pérdida de precisión. Otras, no lo es. Veamos algunos ejemplos:

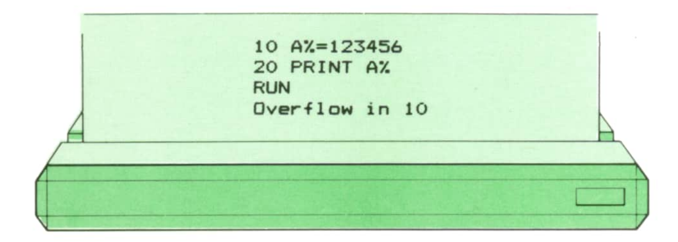
```
10 LET A!=123.456700000000
20 LET B!=123.456723451678
30 LET C!=123.456789012345
40 LET D%=15.000
50 LET E%=15.234
60 LET F%=15.55
70 PRINT A!
80 PRINT B!
90 PRINT C!
100 PRINT D%
110 PRINT E%
120 PRINT F%
RUN
123.4567
123.4567
123.4568
15
15
16
```

Podemos ver que este programa asigna a las variables A! y D% valores que pertenecen a un tipo más amplio que el de ellas, pero que al convertirse a éste resultan ser totalmente equivalentes a los que tenían. Sin embargo, esto no es siempre posible: los valores asignados a las variables B!, C!, E% y F% no pueden convertirse al tipo de éstas sin pérdida de precisión. En este caso la conversión se realiza de todos modos, sustituyendo el valor asignado por aquél que pertenezca al tipo de la variable, que sea lo más próximo posible al que se nos ha dado. Puede verse que a veces esto significa redondear hacia arriba, mientras que otras obliga a redondear hacia abajo. Por convenio, 2.5 se redondeará hacia 3, mientras que 2.499999 se redondea hacia 2.

La conversión de tipo se realizará igualmente, si es necesario, cuando asignemos un valor a un elemento de una serie o tabla de datos:

```
10 DIM X%(10)
20 FOR I=1 TO 10
30 LET X%(I)=I: NEXT I
40 LET X%(3)=5.5
50 FOR I=1 TO 10
60 PRINT X%(I);: NEXT I
RUN
1 2 6 4 5 6 7 8 9 10
```

A veces, sin embargo, la conversión de tipo no puede realizarse, como cuando intentamos asignar a una variable entera un número entero que rebasa la precisión posible en la representación interna de los enteros (véase el capítulo 6) y, por tanto, tiene que representarse en uno de los tipos reales. En tal caso, obtendremos un mensaje de error. Recuérdese que, en la mayor parte de los intérpretes y compiladores BASIC, los números enteros deben estar comprendidos entre -32768 y 32767. Veamos un ejemplo:



```
10 A%=123456
20 PRINT A%
RUN
Overflow in 10
```

Hemos intentado asignar a una variable entera (A%) un valor (123456) que no está comprendido entre -32768 y 32767 y el intérprete lo ha rechazado.

En PASCAL, las normas de conversión de tipos son muy semejantes a las de BASIC, aunque con algunas diferencias. En primer lugar, como en BASIC, no es posible asignar un valor de tipo literal (CHAR o STRING) a una variable de tipo numérico (INTEGER o REAL), ni tampoco al revés. En cambio, sí suele ser válido guardar un dato CHAR en una variable STRING, como en el programa siguiente:

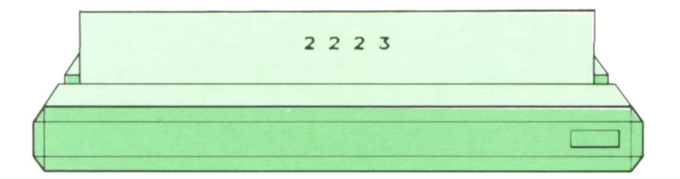
```
program EJEMPLO;
var
  a: char;
  b: string;
begin
  a:='A';
  b:=a;
end.
```

Lo contrario no es posible, como es natural, a menos que el dato STRING que asignamos a una variable de tipo CHAR tenga un solo carácter.

De igual manera, la conversión del tipo entero al real siempre es posible, pero lo contrario no siempre lo es. Además, si en PASCAL queremos asignar un dato con decimales (como 2.6) a una variable de tipo entero, debemos especificar cómo deseamos que se haga la conversión: truncando el valor (con lo que se convertiría en 2) o redondeándolo (con lo que iría a parar a 3). Para ello existen las funciones TRUNC (truncar) y ROUND (redondear). Veamos un ejemplo:

```
program EJEMPLO1;
var
  a, b, c, d: integer;
begin
  a:=TRUNC(2.3);
  b:=ROUND(2.3);
  c:=TRUNC(2.6);
  d:=ROUND(2.6);
  writeln (a,' ',b,' ',c,' ',d);
end.
```

cuya ejecución da el siguiente resultado:



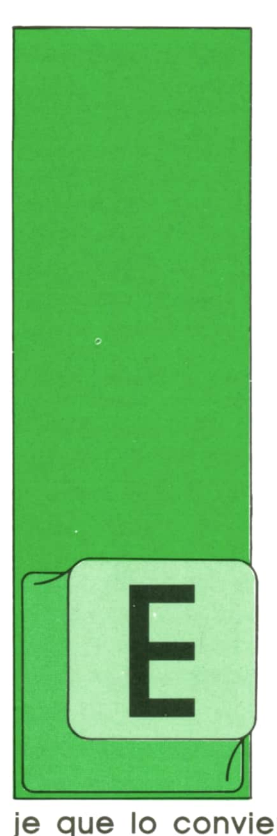
Hemos visto que el problema de la conversión no se presenta en los programas escritos en APL, en los que, independientemente de lo que haga el intérprete con los datos, desde el punto de vista del programador no existen más tipos que los números y los caracteres. De igual manera, en la mayor parte de los programas BASIC se utilizan sólo variables reales en precisión normal, suficiente para muchas aplicaciones, por lo que tampoco es preciso tener en cuenta el problema de la conversión. Este surge sólo cuando se desea aquilatar la ocupación de espacio, la velocidad de cálculo o la precisión aritmética de los programas y se utilizan también variables enteras o en doble precisión.

En PASCAL, sin embargo, el problema de la conversión de tipos puede surgir con más frecuencia, puesto que la necesidad de declarar todas las variables lleva al programador a utilizar distintos tipos de datos en sus programas. No obstante, en muchos programas no llegará siquiera a presentarse, pues las variables enteras suelen utilizarse como contadores, a los que siempre se asigna valores estrictamente enteros, mientras que las variables reales tienden a mezclarse en las operaciones con otras del mismo tipo.

A pesar de todo, es preciso tener presente la posibilidad de que tengan lugar conversiones automáticas de tipos, pues esto puede aclarar algunos efectos que, de no tenerlas en cuenta, resultarían casi inexplicables para el programador principiante.

APLICACIONES

TRATAMIENTOS DE TEXTO: OTROS PROGRAMAS





Personal editor

EL Personal Editor es un programa diseñado para llevar a cabo la edición de textos. Su principal característica es la sencillez de manejo que permite un rápido aprendizaje que lo convierte muy pronto en una aplicación productiva.

Al arrancar el programa se visualiza una pantalla de las más simples que puede uno encontrarse. Una línea en la parte inferior permite introducir comandos. En la parte superior de la pantalla se observan las marcas de principio y final del fichero. Por supuesto, nada más arrancar estas líneas están juntas y sin ningún contenido entre ellas.

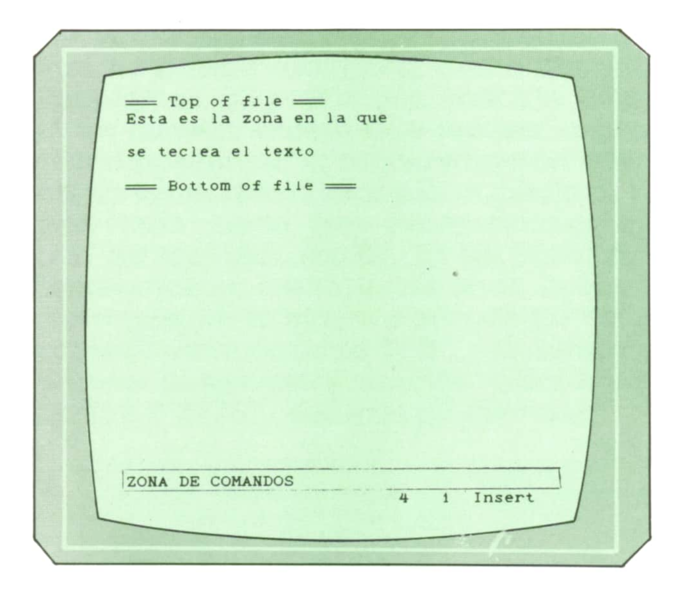




Fig. 1. Pantalla del Personal Editor.

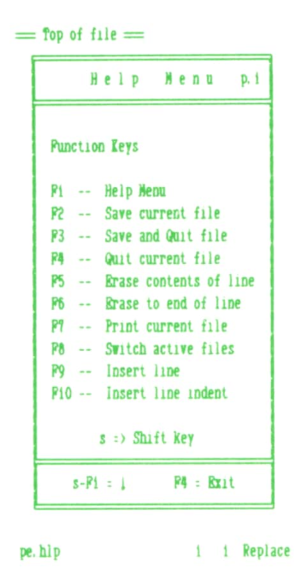
La tecla ESC permite el salto de la línea de comandos a la función de introducción de texto. En esta segunda función, el Personal Editor funciona como si se tratara de una hoja de papel; uno va escribiendo y debe cuidar dónde quedan las palabras para que al imprimir (si fuera ese el caso) cuadren bien.

Los comandos principales realizan las funciones de llamada a un fichero **(e nombre de fichero),** archivo de un fichero en uso **(file nombre de fichero),** salida del programa **(quit),** etc. Además, las teclas de función incorporan estos comandos, lo que facilita el trabajo. Una interesante posibilidad es incorporar al fichero de definición del teclado algunas asignaciones de cadenas de caracteres a teclas concretas. Esto en algunas aplicaciones facilitará indudablemente el trabajo.

El Personal Editor permite la edición de varios textos simultáneamente. Esta posibilidad, dota al programa de gran flexibilidad porque permite componer a partir de un texto ya elaborado otro de similares características o contenido. El fichero de ayuda utiliza esta opción.

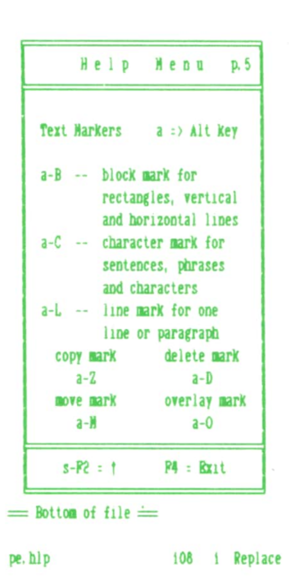
Al pulsar F1, el Personal Editor llama al fichero de ayuda (PE.HLP) y lo edita. Con esta visualización en pantalla se dispone de una ayuda bastante completa de las principales funciones del cursor, las teclas de función, así como las teclas que realizan las funciones básicas del programa (Alt + TECLA).

En la figura 2 se observa una de las pantallas del menú de ayuda. La descripción de cada función está bien realizada en el manual del programa. Sin embargo, destacaremos la posibilidad de partir líneas (Alt-S) y unirlas (Alt-J), que permite recolocar fácilmente partes de texto y agruparlos a gusto de la persona que edita. Es posible trazar cuadrados y relle-





Primera pantalla del menú de ayuda que se encuentra en el fichero Pe.hlp.





Ultima pantalla de Pe.hlp. Aparece la marca de Bottom of file.

nar bloques con caracteres. Las funciones de marca, copiado, movimiento y borrado de bloques se efectúan con gran sencillez.

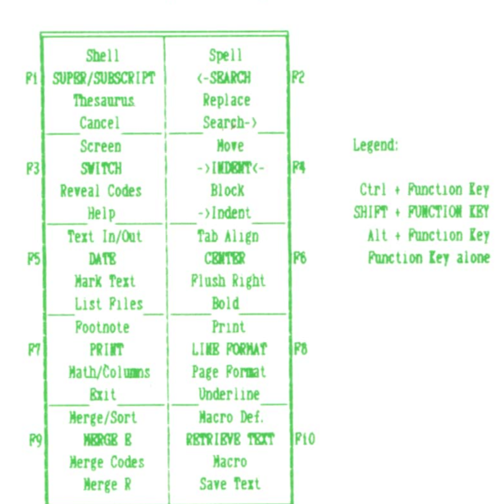
Por otra parte, el Personal Editor requiere que se sitúe un carácter de retorno de carro y alimentación al final de cada línea. En ello se asemeja a una máquina de escribir que al llegar al final de una línea necesita una alimentación "manual" de línea. Es posible no incluir este salto de línea, pero entonces se requiere hacer uso de las teclas de movimiento de cursor. Cuando no se va a emplear un texto escrito en Personal Editor en otro procesador de texto, esto no importa. Sin embargo, si se va a utilizar en otro programa que trate textos, la aparición de esos caracteres de salto de línea puede dificultar enormemente nuestra labor. El programa podría no llegar a reformatear todo el texo, ya que al encontrar un salto de línea supondría fin de párrafo. Si esto ocurre en todas la líneas, el texto quedaría como estuviera. Este aspecto debe cuidarse al escribir con el Personal Editor.

Por último, vamos a reseñar un aspecto de gran interés para los posibles usuarios del Personal Editor. A la par de ser un programa enormemente fácil de manejar, aunque con un menor número de funciones que otros, sus requerimientos de memoria son pequeños. Con sólo 128 Kbytes de memoria ya se puede trabajar con textos de tamaño aceptablemente grande.



Word Perfect

El Word Perfect es uno de los programas de tratamiento de texto más completos y potentes del mercado. La forma de guiarse a través del programa no es mediante menús, como en otros procesadores. En el Word Perfect se dispone de una retícula para situar en torno a las teclas de función. En ella pueden verse las posibilidades que se obtienen al pulsar cada te-





Retícula de las teclas de función del Word Perfect.

cia directamente o combinada con SHIFT, ALT o CTRL, con lo que se tienen 40 funciones visibles en todo momento. Además, una vez seleccionada una de estas funciones, se visualizan en pantalla unos menús de guía para la función en cuestión.

El Word Perfect se completa con unos programas adicionales, de entre los que destaca el potente diccionario que facilita enormemente las correcciones ortográficas.

La flexibilidad y la potencia son posiblemente las dos principales características del Word Perfect. Respecto a la flexibilidad, podemos citar las múltiples opciones seleccionables para dirigir la impresión de textos o documentos, las amplias posibilidades para definición de formatos de pantalla, macros para textos repetitivos y todo un conjunto encaminado a permitir al usuario configurar un procesador a la medida de sus necesidades.

La potencia del Word Perfect se refleja principalmente en la forma de tratar los textos. En cuanto a escritura y visualización en pantalla, destaca el hecho de que las partes de un texto que van en negrilla (y que en otros procesadores se enmarcan entre dos comandos) aparecen resaltadas en pantalla, con lo que se tiene el texto tal como se verá al imprimir. Igual ocurre para las partes de los textos subrayadas. Otra importante función del Word Perfect que potencia su capacidad es la posibilidad de generar un "SPOOL" de impresión, es decir, ir acumulando toda una serie de ficheros a imprimir y desentenderse de ellos. Así se puede seguir trabajando y cada vez que se finalice la impresión de un fichero, será el propio Word Perfect el que tomará el siguiente de los acumulados para imprimir.

El Word Perfect permite también tener varias pantallas con varios textos de forma simultánea, lo que capacita al usuario para combinar documentos distintos con gran flexibilidad.

```
Una pantalla sencilla
pulsando F8 podemos crear líneas continuas:


O lineas subrrayadas.


Pulsando F8 al mismo tiempo que mantenemos la tecla Shift
presionada, se visualiza el menu de ayuda de la parte inferior de
la pantalla.

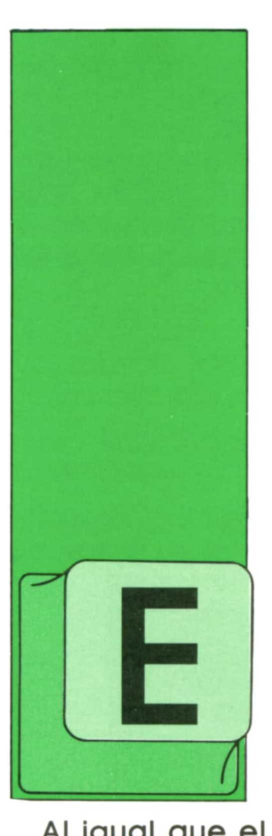

1 Tabs; 2 E-Tabs; 3 Margins; 4 Spacing; 5 Hyphenation; 6 Align Char: 0
```



Pantalla del Word Perfect.

La pantalla del Word Perfect es también muy sencilla. No aparecen menús, salvo que vaya a realizarse una función solicitada con las teclas de función. Con ello, casi exclusivamente con la retícula de ayuda podrían darse los primeros pasos en el manejo del Word Perfect. Sin embargo, para aprovechar este programa en toda su potencia, es necesario leer detenidamente el manual.

Por último, reseñar la reciente aparición de la versión 4 del Word Perfect II con nuevas posibilidades.



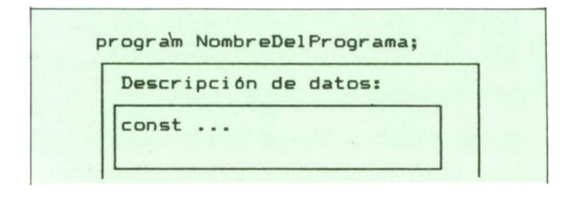
PASCAL

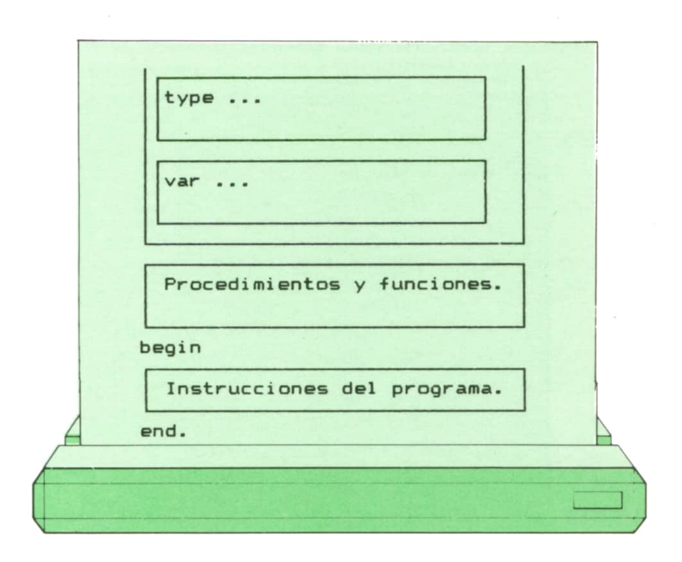
TIPOS DE DATOS A MEDIDA

EN alguna ocasión hemos mencionado que, además de los tipos de datos existentes en PASCAL, es posible crear nuestros propios tipos de datos.

Al igual que el tipo utilizado para manejar números enteros tiene un nombre que lo identifica, INTEGER, los creados por nosotros normalmente tendrán su propio nombre, que puede ser cualquier identificador válido. No obstante, para mayor claridad y siguiendo una norma ampliamente extendida entre los programadores de PASCAL, diferenciaremos los nombres de tipos de los demás de alguna manera; en los ejemplos que hagamos, acabarán siempre con _t; si nuestro ordenador careciera del símbolo de subrayado, o nuestro compilador no lo admitiese, bastará con eliminarlo.

La definición de tipos se hace en la zona de descripción de datos y antes de la definición de variables. Va precedida de la palabra reservada TYPE, tras la cual se escriben las diferentes descripciones de datos separadas entre sí por punto y coma. Por tanto, la estructura de un programa PASCAL, con lo que sabemos hasta ahora, sería:





Por supuesto, y siguiendo la analogía entre programas y procedimientos, éstos pueden tener a su vez tipos locales para su uso exclusivo; como es lógico, irán definidos entre las posibles constantes y variables locales.

Como los procedimientos READ, READLN, WRITE y WRITELN ya están programados de antemano, no sirven para ser utilizados con los tipos creados por nosotros.

Vamos a ver algunos de los posibles nuevos tipos:



Tipos definidos por enumeración

A menudo se trabaja con datos que no son números ni letras. Por ejemplo, podría ser necesario tener en cuenta el día de la semana para que un programa hiciera o no determinadas cosas.

Para guardar en una variable el día en cuestión, una solución podría ser asociar a cada día de la semana un número: lunes el 1, martes el 2, etc. Entonces podríamos escribir cosas como:

```
if Dia = 7 then
  write ('El día es domingo.');
```

donde Dia sería una variable de tipo INTEGER.

Esto nos obligaría a recordar en cada momento qué números hemos asociado a los posibles valores de nuestros datos.

En PASCAL, sin embargo, es posible crear un nuevo tipo de datos para que se ajusten a nuestras necesidades. Podríamos tener así un tipo (llamémosle DIADELASEMANA), cuyos posibles valores serían LUNES, MARTES, MIERCOLES..., al igual que existe el tipo INTEGER, cuyos posibles valores son... −2, −1, 0, 1,..., o el BOOLEAN con TRUE y FALSE.

Definiríamos entonces el tipo enumerando sus posibles valores, que estarán representados por los identificadores válidos que se deseen. Para ello se escribe en primer lugar el nombre del tipo, que irá seguido de un signo igual y de los identificadores de los posibles valores, separados entre sí por comas y encerrados entre paréntesis. Los tipos así creados, junto con los tipos INTEGER, CHAR y BOOLEAN, se llaman TIPOS ESCALARES. Por ejemplo:

```
program TipoEnumerado;

 const  Maximo = 10;

 type
  DiaDeLaSemana_t =
        (Lunes, Martes, Miercoles,
         Jueves, Viernes, Sabado, Domingo);

  ColorPrimario_t = (Rojo, Verde, Azul);

 var
  Hoy, Dia      : DiaDeLaSemana_t;
  Color,
  ColorPintura  : ColorPrimario_t;
  I,J           : integer;
  C             : char;

begin
```

(No se extrañe el lector si encuentra casualmente ejemplos parecidos de tipos definidos por enumeración en más de un libro sobre PASCAL.) Con esos tipos así definidos se podría escribir:

```
Hoy:=Lunes;
if Dia = Sabado ) and
(Color <> Rojo) then ... ;
```

Hay dos restricciones a la hora de crear estos tipos:

— El número de posibles valores está limitado, dependiendo este límite de cada compilador; normalmente es 256.

— Los posibles valores deben ser exclusivos del tipo que hayamos definido. Así, no sería posible crear el tipo VOCALES formado por los valores 'A', 'E', 'I', 'O', 'U', pues éstos lo son del tipo CHAR.

Como se puede ver, el tipo BOOLEAN sería definible, caso de que no lo estuviera ya, de la siguiente manera:

```
type Boolean=(False,True);
```

Al igual que los tipos CHAR y BOOLEAN, los tipos definidos por enumeración tie-

nen asociados números ordinales y se encuentran ordenados según éstos. Estos números se asignan según el orden en que se hayan escrito los posibles valores al definir el tipo, empezando por cero.

Por ello, es posible utilizar las funciones ORD, PRED y SUCC, que dan respectivamente, el número de orden, el valor anterior y el posterior a uno dado:

ord (Hoy)	valdría 1 si Hoy valiera Martes
pred (Sabado)	valdría Viernes
succ (Color)	valdría Verde si Color valiera Rojo.

y comparar valores:

Lunes < Martes

```
procedure WriteDia (D: DiaDeLaSemana_t);
(* Presenta en pantalla el valor de D *)

 begin
  if D = Lunes then write ('Lunes')
   else if D = Martes then write ('Martes')
    else if D = Miercoles then write ('Miércoles')
     else if D = Jueves then write ('Jueves')
      else if D = Viernes then write ('Viernes')
       else if D = Sabado then write ('Sábado')
        else write ('Domingo')
 end;


procedure LeeColor (var C: ColorPrimario_t);
(* Lee de teclado un color *)

   var Letra: char; Ok: boolean;
 begin
  (* Pedir letra del color: *)
  write ('Color? (R,V,A) ');
  repeat
    readln (Letra);
    Ok:= (Letra='R') or (Letra='V') or (Letra='A');
    if not Ok then writeln ('Letra no válida.')
  until Ok;

  if Letra = 'R' then C:= Rojo
    else if Letra = 'V' then C:= Verde
     else C:= Azul
 end;
```

valdría TRUE, pues ord (lunes) es menor que ord (Martes). (Recordemos, una vez más, que en PASCAL nunca se pueden mezclar tipos.)

Por esto mismo, también es posible usar variables de tipo definido por enumeración para control de las estructuras FOR:

for Dia:= Lunes to Viernes do ...
for Color:= Azul downto Rojo do ...

Como los procedimientos que ya conocemos para presentar cosas por pantalla no nos sirven, tendremos que escribirlos cuando sea necesario. Los parámetros que se pasan a un procedimiento pueden ser del tipo que se quiera, incluyendo, por supuesto, a los creados por nosotros:

En el procedimiento de lectura, se ha pasado el parámetro por nombre, pues lo que queremos es alterar el valor de una variable según la letra tecleada. Con esos procedimientos podríamos escribir cosas como:

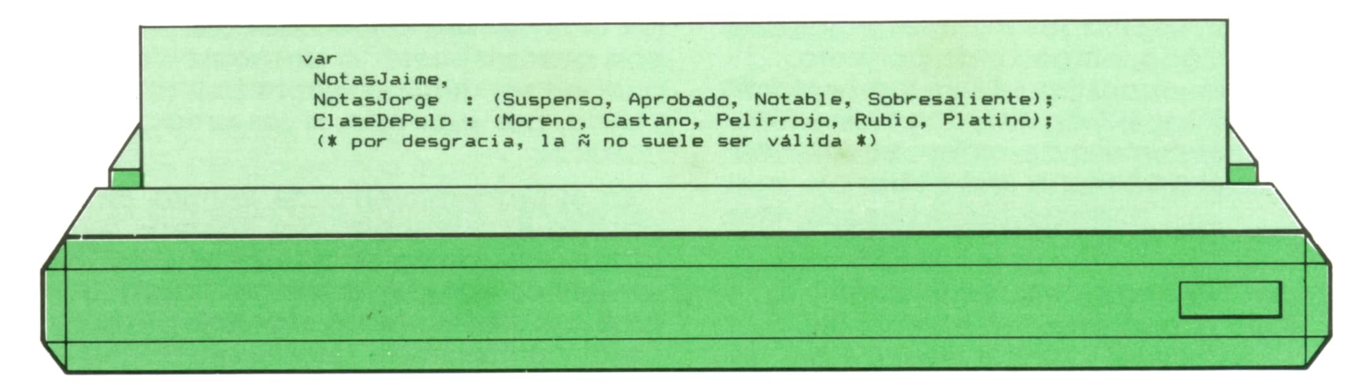
WriteDia (succ(Hoy));
LeeColor (ColorPintura);



Descripción de un tipo sobre la marcha

Cuando sólo se va a hacer referencia a un tipo en un punto de un programa, es posible definirlo al mismo tiempo que se definen las variables de ese tipo, sin necesidad de darle un nombre declarándolo previamente. Para ello basta con poner, al definir la variable, la descrip-

ción de tipo que se habría utilizado si se hubiera declarado de la forma habitual:



```
var
 NotasJaime,
 NotasJorge   : (Suspenso, Aprobado, Notable, Sobresaliente);
 ClaseDePelo : (Moreno, Castano, Pelirrojo, Rubio, Platino);
 (* por desgracia, la ñ no suele ser válida *)
```

Normalmente, no es muy aconsejable proceder así por motivos de claridad; por otra parte, si en un procedimiento hubiera que utilizar parámetros de alguno de esos tipos, no tendríamos más remedio que darles previamente un nombre, pues la definición sobre la marcha no está permitida en las listas de parámetros.

Aunque el ejemplo consta únicamente de tipos definidos por enumeración, se puede definir sobre la marcha el tipo que se quiera.



Tipos subrango

Ya se vio en su momento cómo se podía definir una variable de tipo subrango de los tipos CHAR e INTEGER.

Con los otros tipos escalares es posible también definir subrangos. Por ejemplo, estando el tipo DiaDeLaSemana_t ya declarado podríamos tener:

```
var
  DiaDeCurre: Lunes..Viernes;
```

con lo que indicamos al compilador que la variable DiaDeCurre sólo puede tener valores comprendidos entre Lunes y Viernes. El subrango se describe, por tanto, poniendo los valores extremos separados por un par de puntos, de manera que el primero sea menor que el segundo.

Aunque en el ejemplo hemos escrito el subrango al definir la variable, normalmente será más conveniente crear primero el tipo subrango y luego utilizarlo al describir las variables:

```
program TipoSubRango;

type
 DiaDeLaSemana_t =
             (Lunes, Martes, Miercoles,
             Jueves, Viernes, Sabado, Domingo);

 DiaLaborable_t  = Lunes..Viernes;
 Mayuscula_t     = 'A'..'Z';
 Minuscula_t     = 'a'..'z';

var
 DiaDeCurre: DiaLaborable_t;
 Inicial    : Mayuscula_t;
 ...
```

Los tipos subrangos en el fondo siguen siendo del tipo que les da origen y por

ello SI se pueden mezclar entre sí y con el tipo escalar asociado si éste es común a ellos. Por ejemplo, si tenemos:

```
var
    A:   1..10;
    B:   0..30;
    C: 10..30;
```

el tipo original asociado a A, B y C es INTEGER y por ello las instrucciones

```
    A:= B;
    write (B+C);
    C:= B;
```

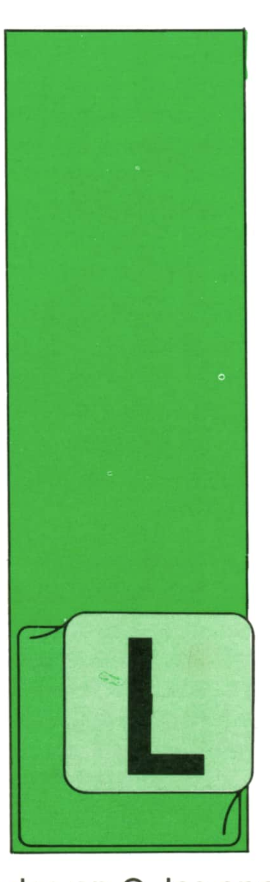
son correctas, aunque al correr el programa se podría producir un error al ejecutar, por ejemplo, A:=B si B tuviera un valor no aceptable para A.

Además de servir para controlar que no se almacenan valores erróneos en las variables, a veces se obtiene un ahorro de memoria, pues el tamaño de la porción necesaria para guardar datos con valores restringidos puede ser menor.

NOTA: La mayoría de los compiladores tienen la posibilidad de activar o no la propiedad de controlar si los valores que se guardan en variables del tipo subrango están dentro de los límites permitidos; para salir de dudas, es menester consultar el manual del compilador.

OTROS LENGUAJES

LENGUAJE C: TIPO DE DATOS, OPERADORES Y EXPRESIONES



A diferencia entre *variables* y *constantes,* como las variables deben *declararse* antes de utilizarse en un programa escrito en C, los diferentes tipos de datos reconocidos en C, los operadores y las expresiones como combinación de variables y constantes.



Variables y constantes

Tanto las variables como las constantes en un lenguaje de programación, son datos que el programa utiliza para realizar una serie de tratamientos con ellos.

Ciertos datos se preseleccionan antes de la ejecución de un programa, manteniendo sus valores inalterables durante la misma, por lo que a dichos datos se les conoce como *constantes.*

Por el contrario, existen otros datos que pueden variar a lo largo de la ejecución del programa, llamándose a éstos *variables.*

El nombre de una variable tiene ciertas restricciones. Veamos esto. Cualquier variable está formada por una serie de letras y dígitos en secuencia, pero debemos tener muy claro siempre que el primer carácter de comienzo de una variable debe ser una letra. Nombres de variables válidos pueden ser:

```
area
h54
juan_ana
```

Por el contrario, no podrán utilizarse para variables nombres tales como:

```
5tuyo
juan.ana
```

Las letras mayúsculas y minúsculas son diferentes en el lenguaje de programación C. Se utilizan tradicionalmente las minúsculas para los nombres de variables y las mayúsculas para nombres de constantes.

Solamente los ocho primeros caracteres de un nombre de variable son significativos, aunque pueden utilizarse en número superior.

Finalmente diremos que en C, como en todos los lenguajes de programación, existen *palabras reservadas* que no podrán utilizarse como nombres de variables. Tal es el caso de las siguientes palabras reservadas:

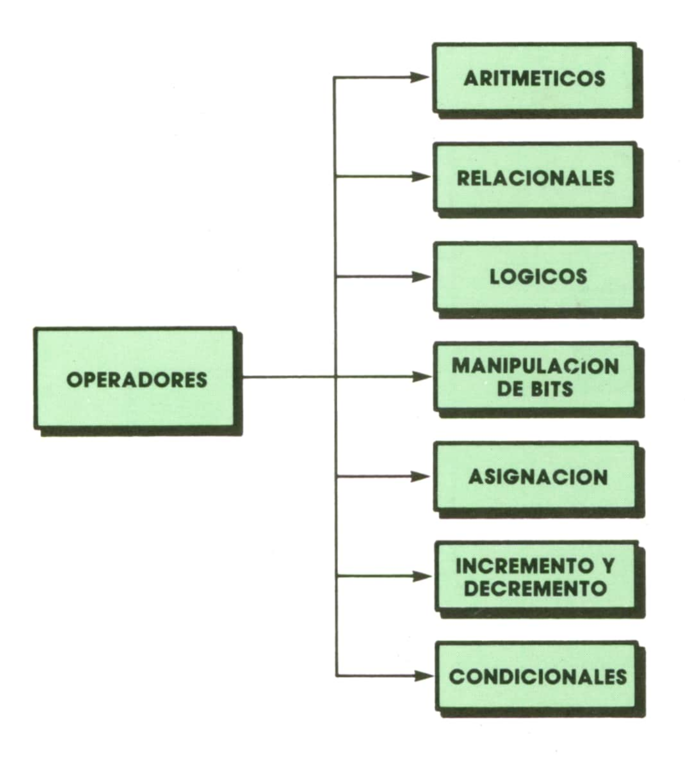
auto	break	case	char	continue
default	do	double	else	entry
enum	extern	float	for	goto
if	in	long	register	return
short	sizeof	static	struct	switch
typedef	union	unsigned	until	void
while				



Datos: sus tipos

Dentro del lenguaje de programación C los tipos de datos se clasifican en:

— *char*
— *int*
— *float*
— *double*




Operadores.

El tipo *char* almacena en un byte (ocho bits) un carácter que corresponde al juego de caracteres empleado en su instalación (ASCII o EBCDIC).

El tipo *int* son enteros (sin parte decimal) que pueden tener signo o no.

Para el tipo *int* existen una serie de calificadores tales como *short, long* y *unsigned.*

El tipo *float* representa un número en punto flotante y simple precisión. Este tipo de datos es utilizado en programas que poseen un gran número de cálculos matemáticos.

El tipo double (doble precisión) utiliza un número doble de bits, es decir, 64. Incorporando los 32 bits adicionales a la mantisa, se consigue un mayor número de cifras significativas, alcanzándose una mayor precisión.



Declaración de variables

Todas las variables en C, al igual que en Pascal, deben ser declaradas antes de su utilización. En todo programa C suministraremos una lista de variables que se utilizarán a lo largo del programa, indicando a qué tipo pertenecen.

Veamos un ejemplo de declaración de variables:

```
int a,b,c;
long int z;
char letra, silaba;
```

Las variables de un mismo tipo pueden reunirse en una misma sentencia o utilizar varias.

En la declaración de variables podemos inicializar éstas a un determinado valor, como, por ejemplo:

```
int p = 75;
char palabra = 'p';
```

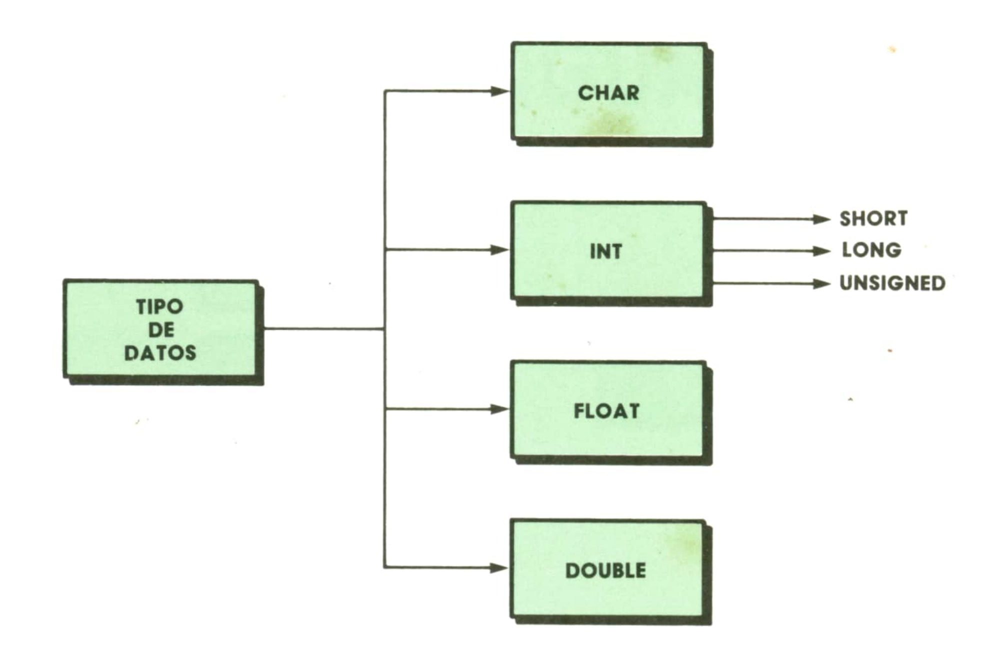
El ejemplo anterior definiría una variable entera con un valor inicial de 75 y una variable del tipo carácter con valor inicial *p.*



Operadores

En el lenguaje C existen los siguientes tipos de operadores:

— *Aritméticos.*
— *Relacionales.*
— *Lógicos.*
— *Manipulación de bits.*
— *Asignación.*
— *Incremento y decremento.*
— *Condicionales.*




Tipos de datos.

Veamos cada uno de ellos.

— *Operadores aritméticos*

Los operadores aritméticos binarios son: + (suma), – (resta), * (multiplicación), / (división) y el operador módulo %.
Si realizamos una división entera de dos números, se truncará cualquier parte fraccionaria.

— *Operadores relacionales*

Los operadores de relación en C son : > (mayor que), < (menor que), >= (mayor o igual) y <= (menor o igual).
Los *operadores de igualdad* son: == (igual) y != (distinto de).

— *Operadores lógicos*

Las conectivas lógicas son : && (and) y || (or).

— *Operadores lógicos para manejo de bits*

Los operadores para la manipulación de bits no son aplicables a operandos del tipo *float* y *double*. Entre los operadores de manipulación tenemos: & (and lógico), | (or lógico), ∧ (or exclusivo lógico), ≫ (desplazamiento a la derecha), ≪ (desplazamiento a la izquierda) y ~ (complemento a uno o negación de bits).
Los operadores lógicos de bits trabajan bit a bit y pueden tratar cada bit independientemente.

— *Operadores de asignación*

El operador de asignación más elemental es el =, que no debe confundirse con el operador == de comprobación de igualdad.

— *Operadores de incremento y decremento*

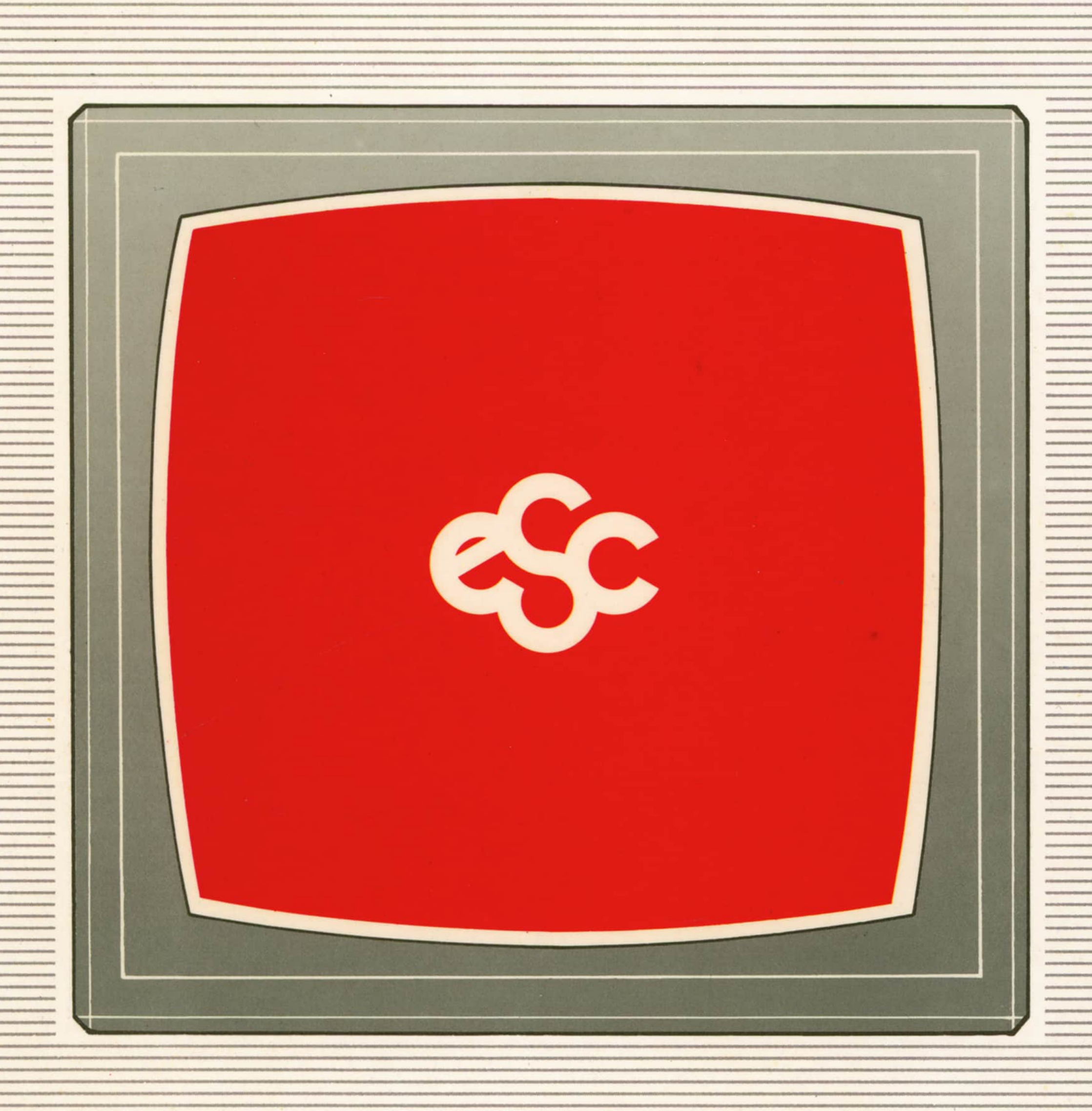
Para incrementar y decrementar en una unidad a un operando el lenguaje de programación C, dispone de dos operadores : ++ (le suma 1 a su operando) y -(le resta 1 a su operando). Estos operadores pueden ir colocados antes de la variable o después de la variable. Dependiendo de la colocación los efectos serán diferentes.

— *Operador condicional*

En C se puede abreviar la sentencia *if-else* tan conocida en otros lenguajes de programación, a través del operador condicional ?:
La forma general de una expresión condicional es:

expresion1 ? expresión2 : expresión3.

Si expresión1 es cierta, la expresión condicional total toma el valor de la expresión2; si, por el contrario, expresión1 es falsa, toma el valor de la expresión3.



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼